Behavioral Service Substitution: Analysis and Synthesis

D I S S E R T A T I O N

zur Erlangung des akademischen Grades

Dr. rer. nat.
im Fach Informatik

eingereicht an der
Mathematisch-Naturwissenschaftlichen Fakultät II
Humboldt-Universität zu Berlin

von
M.Comp.Sc. Jarungjit Parnjai

Präsident der Humboldt-Universität zu Berlin:
Prof. Dr. Jan-Hendrik Olbertz

Dekan der Mathematisch-Naturwissenschaftlichen Fakultät II:
Prof. Dr. Elmar Kulke

Gutachter:
1. Prof. Dr. Wolfgang Reisig
2. Prof. Dr. Holger Schlingloff
3. Prof. Dr. Karsten Wolf

eingereicht am: 29.02.2012
Tag der Verteidigung: 21.03.2013

Abstract

Service-oriented computing has emerged as an acknowledged paradigm for developing complex and non-monolithic systems of different kinds. The paradigm allows different organizations to aggregate several simple *services* to form a composition of communicating entities in the system. Naturally, services in such a system are subject to change and adaptation over time. Service-oriented computing provides the support for service evolution by allowing one service to be *substituted* by other services. From the viewpoint of a single service, the composition of the other communicating services in the system is regarded as *partner*. Substituting one service by another should guarantee to preserve every partner that is *correct* according to a *correctness criterion* of composition in the overall system.

The aforementioned substitution criteria impose difficulties for a service designer, such as a domain expert, to perform analysis and synthesis tasks on service substitution. Due to the complexity of service interaction in the system, it is not straightforward for a service designer to foresee every partner and every substitute for the given service under a specific correctness criterion.

In this thesis, we develop an approach that shall assist a service designer to perform analysis and synthesis tasks on service substitution in a way that every partner of a given service must be preserved under substitution. We model a *control flow* of services that describes the ordering of asynchronously communicating events over an implicit unordered message buffer. We study the *behavioral* aspect of correct interaction between services and concentrate on two variants of *deadlock freedom* as correctness criteria of service composition.

The main contribution of this thesis is an approach for characterizing the set of all substitutes for a given service. The central idea of the approach is to systematically investigate the relationship between a service and all its partners under a given correctness criterion. We employ this relationship to synthesize from a given service its *canonical partner* and its *canonical substitute* with respect to all partners. A service that *refines* the canonical substitute for a given service is regarded as a substitute for the given service if the set of all its partners includes every partner of the given service. With the canonical substitute of a given service, we identify a specific subset of the set of all substitutes for the given service, each of which has exactly the same set of partners as that of the given service.

Parts of the results in this thesis have been established upon previous works on service substitution and correctness of services and their composition. Consequently, we can also combine our results with the related existing techniques to perform more sophisticated analysis and synthesis tasks on service substitution.

Contents

Part I.

Preliminaries

1. Introduction

Nowadays, many computer-based systems are executed in an open environment. Such a system is non-monolithic and can be built from several smaller software components. Each component is capable of processing own *activities* until it reaches the point of *interaction* with another component in the system. In general, the interaction between two components plays an important role for determining how each component proceeds further. The growing complexity of interaction among components inherently makes it difficult for a domain expert to analyze and assess the behavior of an individual component within the system.

Service-oriented computing [Papazoglou, 2003, Papazoglou et al., 2007] has emerged as an acknowledged computing paradigm for developing complex and non-monolithic systems of different kinds. The paradigm allows different organizations to compose several self-contained software components, called *services*, to form an interaction of communicating components in the system. Each *service* offers an encapsulated functionality through a well-defined interface by providing certain *activities* as atomic units of work that it can perform. The separation of functionality and interface allows a service to be developed independently of underlying technologies and concrete hardware platforms.

In general, a service is not designed to execute in isolation. To realize the execution, a service performs either its own activities or interacts with another service typically by exchanging messages in an asynchronous fashion. The *behavior* of a service is described by a partial order of all its activities and distinguished by states of interaction with other services. The service behavior is closely related to the term *communication protocol* or *business interaction protocol* [Papazoglou, 2008a], which specifies the order in which service activities are executed. Another prominent class of services are *Web Services* [Papazoglou et al., 2007, Papazoglou, 2008a]. Web services employ the Internet as the communication medium and open Internet-based standards for realizing the services-oriented computing paradigm.

The interaction of services can be realized with the concept of *composition*. For describing the interaction of services, the terms *choreography* and *orchestration* have been widely used to describe two aspects of composing a more complex service from several services [Peltz, 2003]. *Service choreography* describes the interaction of services from a global perspective of all services. On the contrary, *service orchestration* describes the behavior of a service composition from the point of view of a single service of this composition. Recently, service choreography and orchestration have attracted substantial attentions from both industrial and academic domains. There are a number of service description languages for specifying service orchestration (e. g., WS-BPEL [Alves et al., 2007] and BPMN [OMG, 2009]) and service choreography (e. g., WS-CDL [Chinnici et al., 2003], Let's dance [Zaha et al., 2006], and BPEL4Chor [Decker et al., 2008]).

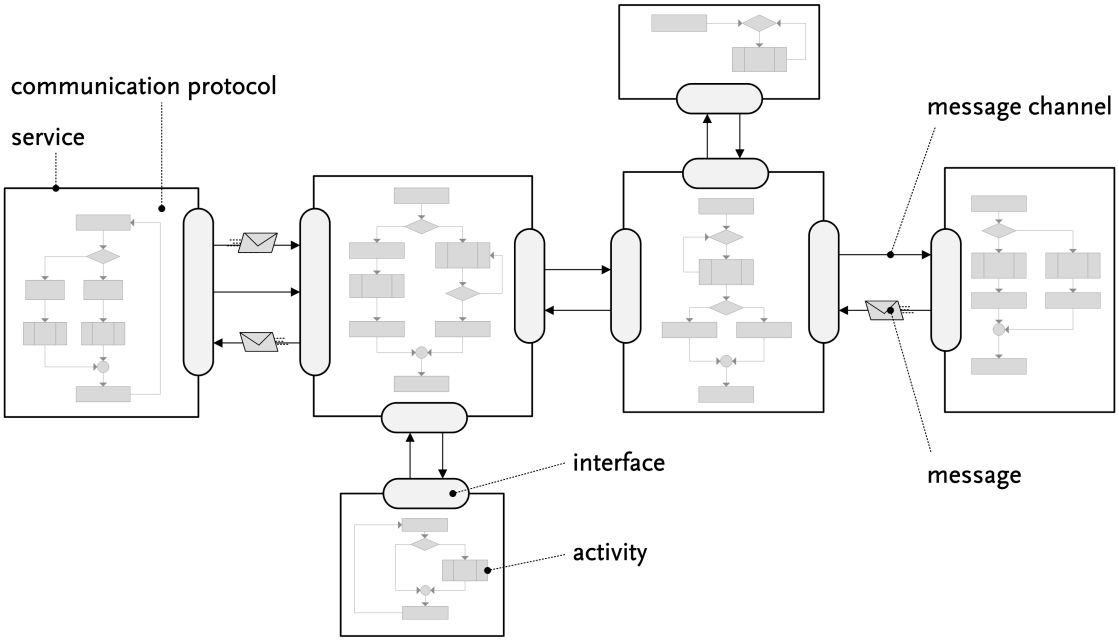


Figure 1.1.: Illustration of services and their composition.

The correctness of service interaction guarantees that one service interacts correctly with others and is defined by the *compatibility* between services. The compatibility of services has been studied from various aspects [Papazoglou, 2008a]. At least four different aspects of service compatibility are described in the literature; *syntactical compatibility*, *behavioral compatibility*, *semantical compatibility*, and *non-functional compatibility*. Syntactical compatibility focuses on the service interface and ensures that message types between services must be compatible [e. g., Dong et al., 2004]. Behavioral compatibility guarantees an absence of behavioral errors, such as deadlocks and divergences [e. g., Bordeaux et al., 2005, Lohmann et al., 2007a, Dumas et al., 2008]. Semantical compatibility ensures correct interpretation of exchanging messages and their contents [e. g., Traverso and Pistore, 2004, Mrissa et al., 2007, Sycara et al., 2011]. Finally, non-functional compatibility guarantees some quality parameters of services, such as security requirements and performance [e. g., Deora et al., 2006, Yu et al., 2007].

Systems naturally evolve over time due to various circumstances such as a change of requirement or regulation, an improvement of functionality, and an adaptation of execution parameters. Every change, however, requires substantial amount of work for integrating it into the system. Service-oriented computing provides the support for service evolution by allowing one service to be *refined* into a new service and *substituted* by the refined service [Papazoglou, 2008b]. The idea of abstracting from underlying technologies enables the comparison of two services according to a given criterion, therefore, makes it possible to reason about substituting one service by another. From the viewpoint of a single service, other communicating services in the system together are regarded as its *partner*. Two partners should be compatible under a given correctness criterion of

service interaction. Substituting one service by its refined service should guarantee to preserve every *compatible* partners of the original service. In the situation where not every compatible partner is a partner of interest, substituting one service by its refined service should guarantee to preserve only selected (possibly non-compatible) partners of the original service. One service is a *compatible substitute* for another service with respect to a given substitution criterion if the preservation of those partners is guaranteed under substitution.

The aforementioned substitution criteria impose difficulties for a service designer, such as a domain expert, to perform analysis and synthesis tasks on service substitution. The substitution of two services is decided by comparing their compatible partners, rather than their own semantics. In asynchronous setting, services communicating with others in asynchronous fashion and it is not trivial to identify semantics of its compatible partners from its own structure. Generally, there are more than one possible compatible partners for a given service under a given correctness criterion. Each service allows its compatible partner to perform non-communicating activities, as a result, the set of all its compatible partners of is possibly infinite. Either a slight refinement of the original service behavior may drastically change the set of compatible partners or a complete reordering of communicating activities may preserve exactly the same set of compatible partners. Therefore it is not straightforward for a service designer to foresee every compatible partner of a service when performing substitution.

To systematically support various analysis and synthesis tasks on service substitution, a *formal framework* that guarantees the correctness under substitution is required.

An analysis task regarding service substitution involves checking whether one service is a substitute for another according to a correctness criterion. Usually, the design of a substitute contains flaws. *Testing* [Myers and Sandler, 2004] is an empirical method which observes an output of the system from given inputs and compares the output to expected result. The purpose of the testing process is to detect the presence of errors or flaws within the system; however, not their absence. As an alternative to testing, *model checking* [Clarke and Schlingloff, 2001, Clarke et al., 2001] has been introduced to automate the verification of the system in a rigorous way. Nevertheless, verifying the entire system interaction using model checking techniques is not a feasible option to guarantee correctness under substitution, as a single service does not have absolute control of its environment and over the overall collaboration of the system. Therefore, the analysis task requires a formal analysis technique for assessing whether one service is a substitute for another service. In case a design flaw of a new service for substitution is detected, another challenging task involves how to systematically repair the detected flaw towards the correctness under service substitution.

An alternative method to ensure correctness of service substitution is *by construction*. This method aims to avoid design flaws at the early phase of development by synthesizing from a given service its substitute. By doing so, we can enforce the correctness in the design process before implementation. Incrementally the service model of a synthesized substitute is refined further either manually or automatically towards its implementation. This task requires a proper technique for synthesizing from a given service a new service that is a substitute for the given service.

1.1. Problem Statement

In this thesis, we develop an approach that shall assist a service designer to perform analysis and synthesis tasks on service substitution in a way that every or selected partner(s) of a given service must be preserved under substitution.

The goal of this thesis is to investigate the three-dimensional problem space related to service substitution as illustrated in Figure 1.2.

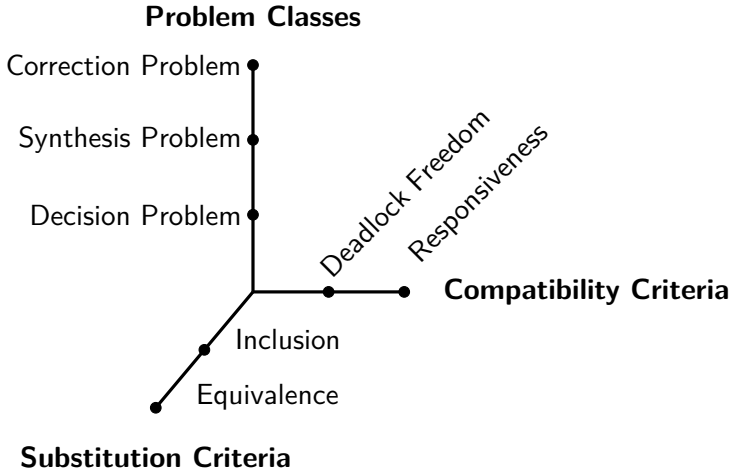


Figure 1.2.: Illustration of the problem space investigated in this thesis

1. **Compatibility Criteria** : The first dimension of the problem space focuses on the behavioral aspect of compatibility of service composition by addressing:
   a) **Classical Deadlock Freedom** guarantees that the services must interact with each other in a deadlock-free manner. This means the service composition will never get stuck in a deadlock state, a global non-final state in which none of participating services can perform any further action.
   b) **Responsive Deadlock Freedom** or **Responsiveness** guarantees that in the system must interact with each other in a deadlock-free manner and responsive manner. It requires that each participating service in the composed system never perform an unbroken sequence of own internal activities without responding to a request from the other service in the composed system.

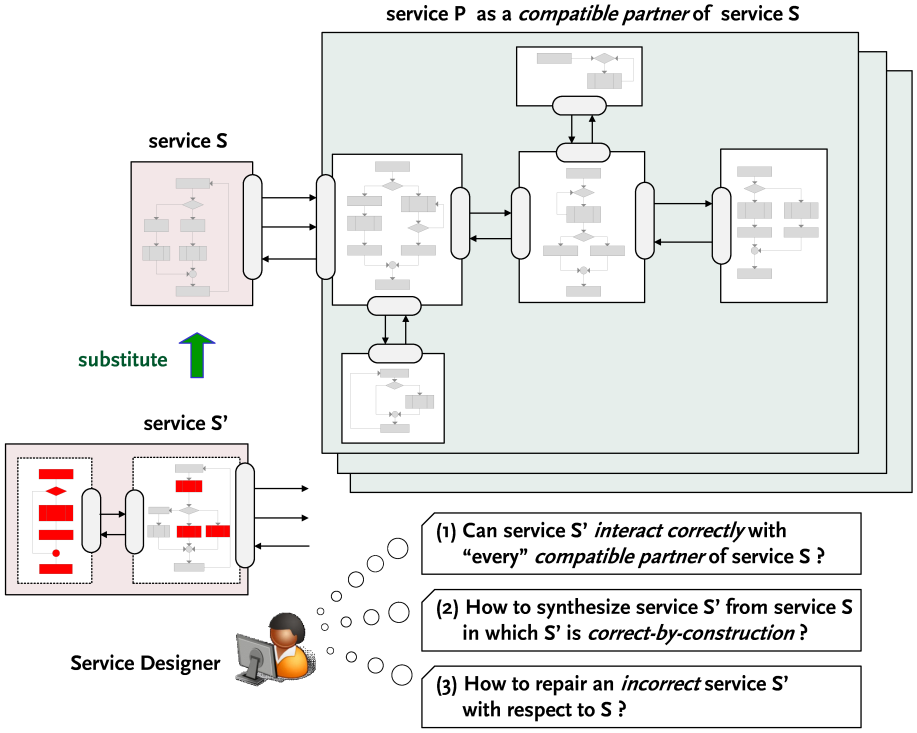
   From a viewpoint of a single service, the other communicating services in the system together constitute a *partner*. Each compatibility criterion defines a *compatible partner* of a service as a partner that interacts with its partner in a compatible manner.

2. **Substitution Criteria** : The second dimension of the problem space focuses on the substitution criterion which guarantees that *every* or *selected* partner(s) of a service to be substituted must be preserved under substitution. Two major substitution criteria are investigated in this thesis :

a) **Inclusion** : substituting service S by a service S' must preserve *at least* every or selected partner(s) of S. This substitution criterion guarantees that the set of all compatible partners of S' must include either all compatible partners of S, or selected compatible partners of S, or selected possibly incompatible partners of S.

b) **Equivalence** : substituting service S by a service S' must preserve *exactly* every or selected partner(s) of S. This means, services S and S' must have exactly the same set of all or selected compatible partners. In contrast to the inclusion substitution criterion, the equivalence substitution allows the reverse direction of service substitution, that is, substituting service S' by service S.

Service S' is a *substitute* for service S whenever service S' can substitute service S under a given substitution criterion.



3. **Problem Classes** : The third dimension of the problem space focuses on three classes of problems that a service designer, e. g., a domain expert, usually encounters during the design phase. These problem classes are described as follows.

a) **Decision Problem** : This problem class addresses the *correctness by verification* for service substitution. Given services S and S', how to systematically *decide* whether or not service S' is a substitute for service S ?

b) **Synthesis Problem** : This problem class addresses the *correctness by construction* for service substitution. Given service S, how to systematically *synthesize* a service S' from service S which is *correct-by-construction* ?

c) **Correction Problem** : This problem class addresses the *error correction* for service substitution. Given a service S and a service S' that is not a substitute for S, how to systematically *repair* S' such that the result is a substitute for S ?

In this thesis, we investigate each point in the three-dimensional problem space from the *behavioral* aspect of correctness criterion between services and assume that other aspects, such as functional, non-functional, and semantical aspects, are not violated when performing substitution. We model a *control flow* of *stateful* services that describes the ordering of asynchronously communicating events over an implicit unordered message buffer. This means, we focus on the *communication protocol* which specifies the order in which either asynchronously communicating events and non-communicating events are executed. The results of this thesis shall provide analysis and synthesis techniques as well as tools to tackle each individual point in the problem space illustrated by Figure 1.2.

1.2. Contributions of this Thesis

The main contribution of this thesis is an approach for characterizing the set of all substitutes for a given service under a given criterion as described in Figure 1.2.

Characterization of all Substitutes under Inclusion

For each compatibility criterion, we characterize the set of all compatible substitutes for a given service using the concept of maximal element of a preordered set of compatible partners w. r. t.the inclusion preorder relation. A maximal compatible partner has several distinguished properties. With these, we can reduce the problem of characterizing the set of all compatible substitutes for a given service under inclusion to the problem of characterizing the set of all compatible partners of its maximal compatible partner.

To realize this result, we propose an algorithm to synthesize a maximal compatible partner from a given service S. This synthesize algorithm is based on an approach called *Operating Guidelines* approach to characterize the set of all deadlock-free partners [Massuthe et al., 2005, Massuthe and Schmidt, 2005, Lohmann et al., 2007a, Massuthe, 2009] and the set of all responsive partners [Lohmann, 2010]. The approach provides an algorithm to synthesize an artifact called operating guidelines which is a finite structure that can compactly represent the set of all compatible partners of a given service via a matching relation. To this end, we propose to synthesize an operating guidelines of a synthesized maximal partner of S. Then the problem of deciding whether or not the refined service S' can substitute the original service S under inclusion can be reduced to the problem of whether or not S' satisfies the matching relation of Operating Guidelines of a maximal compatible partner of S.

The algorithm can be used to synthesize a maximal partner that represents the set of selected (possibly incompatible) partners of S. When combining with the Operating Guideline approach, the problem of deciding whether or not S' can substitute S under selective inclusion can be reduced to the problem of whether or not S' satisfies the matching relation of the operating guidelines of the synthesized maximal partner of S.

Canonical Representative of all Compatible Substitutes and Relationship with Failures Refinement

We systematically investigate the relationship between a service S and all its compatible substitutes under each compatibility criterion via its maximal partner. For each compatibility criterion, we employ this relationship to identify a closure operation on the service S which maps S onto its own equivalence class. Such an operation can be applied canonically to S multiple times without changing the result beyond the initial application.

We realize the closure operation by synthesizing a maximal compatible partner of a maximal compatible partner of service S. The synthesis algorithm produces from service S one of its equivalent substitute containing a maximum number of traces among all other substitutes for S. To this end, the synthesized equivalent substitutes can be regarded as a canonical representative all of compatible substitutes for S. We employ the respective failures (stable failures refinement in case of deadlock freedom and responsive failures refinement in case of responsiveness) as a refinement relation of the synthesized equivalent substitutes for S. We show that a service S' refines the synthesized equivalent substitutes for S under the respective failures whenever S' is a substitute for S under the respective inclusion.

In case of deadlock freedom, Stahl [2009] has shown a result indicating a coincidence between stable failures refinement [see e. g., Hoare, 1985b, Roscoe, 1998] and our deadlock freedom inclusion (called *accordance* in Stahl [2009]) for synchronously communicating services. In our asynchronous setting, a service communicates asynchronously with its compatible partner via an implicit, bounded, and unordered message buffer. We will illustrate with some examples (cf. Figure 4.1 and Figure 4.4 for instances) that there is no immediate coincidence between our deadlock freedom inclusion with stable failures refinement or any other classical refinement relation that is known to us. Nevertheless, we employ stable failures as a refinement relation on the synthesized equivalent substitutes for a given service S and show that the problem of deciding whether or not the refined service S' can substitute the original service S under deadlock-free inclusion can be reduced to the problem of whether or not S' refines the synthesized equivalent substitute for S under stable failures.

In case of responsiveness, none of the classical refinement relations that are known to us can be employed as refinement relation of responsiveness inclusion. In this thesis, we propose an extension of stable failures semantics called *responsive failures* semantics. The responsive failures semantics extends the stable failures semantics by treating a *livelock* of non-communicating events similarly to a deadlock. Therefore, we ignore a local *livelock* of non-communicating events as long as it is possible to perform either an asynchronously communicating event or a successful terminating event. To this end, we employ the responsive failures as a refinement relation of the synthesized equivalent substitutes for a given service S and show the problem of deciding whether or not the refined service S' can substitute the original service S under responsive inclusion can be reduced to the problem of whether or not S' refines the synthesized equivalent substitute for S under responsive failures.

Characterization of all Substitutes under Equivalence

For each behavioral compatibility criterion, we present a characterization of the set of all equivalent substitutes for a given service. We identify the necessary and sufficient conditions for deciding deadlock-free equivalence of services. To decide if the refined service S' is an equivalent substitute for service S, service S' must satisfy two conditions on the synthesized equivalent substitutes for S. Firstly, service S' must refine the synthesized equivalent substitutes for S under the respective failures (stable failures refinement in case of deadlock freedom and responsive failures refinement in case of responsiveness). Secondly, service S' must cover the relevant knowledge that the synthesized equivalent substitutes for S has about all its compatible partners. We show that service S' satisfies these conditions with respect to service S whenever S' is an equivalent substitute for S under the respective equivalence.

Applicability to Analysis and Synthesis Tasks for Service Substitution

The theoretical results in this thesis enable several applicabilities to analysis and synthesis tasks for service substitution. These applicabilities introduce solutions to the decision problem, the synthesis problem, the correction problem described in the problem space from Figure 1.2. In this thesis, we address the following applications.

– An alternative procedure to decide service substitutability under deadlock-free inclusion and responsive inclusion. Although requiring more pre-processing steps of computation, our procedure requires less steps of computation when deciding service substitution in comparison to existing procedures [Stahl and Wolf, 2009, Stahl and Vogler, 2011, Vogler et al., 2012].

– An alternative procedure to synthesize a *public view* of a given service. We propose to employ the synthesized equivalent substitutes for S as a public view of S. In comparison to existing procedures Wolf [2007a], our procedure requires more pre-processing steps of computation and producing a larger space of public view. Nevertheless, we provide a technique to refine a public view of S into a private view of S which is guaranteed as a compatible substitute for S.

– A procedure to repairing undesirable behavior of the refined service S' with respect to an original service S. We provide a technique based on the Operating Guidelines approach to analyze whether it is possible to repair service S' by removing incorrect behaviors. In case it is possible, we propose a number of edit actions required to modify service S' and guarantee that the modified service is a compatible substitute for S.

– A procedure to synthesize substitutes for a service with selected partners. In case the inclusion substitution criterion is too restrictive, we provide a technique to synthesize a new version S' from the selected partners of S and guarantee that the service S' is a compatible partner with every selected partner of S. The procedure does not require a selected partner to be a compatible partner of S and does not require the set of selected partner to be finite.

– A procedure to integrating selected services for the purpose of finding substitutes for every selected services. We provide a technique to synthesize a new version S' from the selected services and guarantee that the service S' is a compatible substitutes for every selected services. The procedure does not require a selected service to be a compatible substitute of another selected service and does not require the set of selected services to be finite.

– A procedure to synthesize substitutes for a given service by means of transformations. We provide a set of transformation rules, each of which guarantees to preserve either deadlock-free or responsiveness inclusion and/or equivalence under transformation.

The analysis and synthesis techniques that are enabled by the results of this thesis can be combined with the existing analysis and synthesis techniques that are based on the *operating guidelines* approach [see e. g., Lohmann et al., 2007b,a, Lohmann, 2008a, Stahl et al., 2009]. The analysis and synthesis tasks for service substitution can be carried out by combining a number of open source software tools in the tool family *service-technology.org* such as BPEL2oWFN [Lohmann, 2007], Fiona [Massuthe and Weinberg, 2008], Wendy [Lohmann and Weinberg, 2010], Cosme [Lehmann, 2011], Petri Net API [Mennicke, Sura, Waltemath, Lohmann, Gierds, and Znamirowski, 2009] and Rachel [Lohmann, 2008b], with the two tools Maxis [Parnjai, 2011c] and *Evans* [Parnjai, 2011b] developed under the course of this thesis. All related softwares are available for download at `http://service-technology.org/tools`.

1.3. Outline of this Thesis

This thesis is divided into four parts.

Part I : Preliminaries provides the introduction and background for the thesis.

– Chapter 1 introduces the topics and summarizes the contributions of this thesis.

– Chapter 2 introduces the basic notions and formalisms that are used for modeling and verifying the service behavior throughout the thesis. The chapter also reviews related work from the literature.

– Chapter 3 introduces existing techniques for representing sets of services. It then generalizes existing techniques and illustrates how to employ the techniques for characterizing sets of all compatible partners of a given service.

Part II : Theory establishes the fundamental theory for formal analysis of service substitutability and synthesis of substituting services.

– Chapter 4 presents two canonical representatives of all compatible partners of a given service. This chapter illustrates the relationship between stable failures semantics (and responsive failures semantics) with deadlock freedom (and responsiveness).

- Chapter 5 defines, for each compatibility criterion, a maximal element of the substitution preordered set of services. This chapter discusses various properties of the maximal element and establishes a relationship between the set of all compatible partners and the set of all substitutes for a given service via their maximal elements.
- Chapter 6 employs the results from the previous two chapters to characterize, for each compatibility criterion, the set of all equivalent substitutes for a given service.

Part III : Applications illustrates in Chapter 7 how to apply the theoretical results from Part II to tackle various analysis and synthesis problems related to service substitution. The chapter compares different procedures for deciding service substitutability according to various substitution criteria. The chapter demonstrates how to repair incorrect behavior of services and how to combine the results from Part II with existing techniques to perform more sophisticated task on analysis of service substitutability and synthesis of substituting services.

Part IV : Summary concludes the thesis in Chapter 8 by summarizing contributions, addresses some open problems, and describes possible extensions of the obtained results.

2. Behavioral Service Substitution

This chapter introduces the basic assumptions and notions that will be used throughout the thesis. It presents *service automata* as a formalism for modeling and reasoning about the behavior of services and their composition. For each behavioral compatibility criterion of service composition, the chapter introduces the two substitution criteria that are investigated in this thesis. It concludes with a summary of related work from the literature.

The chapter is organized as follows. Section 2.1 introduces service automata and their related elements. Section 2.2 defines services composition and discusses necessary assumptions. Section 2.3 presents the two behavioral compatibility criteria of service composition that are investigated in this thesis. Section 2.3 introduces the notion of controllers for each compatibility criterion of service composition. Section 2.4 introduces the two substitution criteria that are investigated in this thesis. Section 2.5 discusses related work from the literature. Finally, Section 2.6 concludes the chapter.

2.1. Service Automata

A service consists of an *interface* for asynchronous communication with other services and of a control structure describing its *behavior*. The *interface* describes the syntactic signature of a service as a list of *open* message channels that are exposed to the environment (that is, to other services). For a given service, its interface consists of a set of *Input* and of *Output* message channels (denoted by I and O respectively). The sets of input and output message channels of a service determine a set of communicating events of service. A set of communicating events of a service consists of a set of message receiving events determined by input message channels and a set of message sending event determined by a set of output message channels. A non-communicating event of a service is called an internal event. In this thesis, we do not distinguish between different internal events. A service terminates successfully if it performs a terminating event. The *behavior* of a given service can be described by the order in which various events occur in a non-deterministic fashion. To specify the *behavior* of a service, we employ *service automata*.

Throughout the thesis, we fix a finite set $\mathbb{M}$ of *message channels*, in which $\mathbb{M}$ is partitioned into the set $\mathbb{M}^I \subseteq \mathbb{M}$ of *input message channels* and the set $\mathbb{M}^O \subseteq \mathbb{M}$ of *output message channels*. From $\mathbb{M}$, we can derive the set Σ of all *communicating events*, in which Σ is partitioned into the set $!\Sigma = \{!m \mid m \in \mathbb{M}^O\}$ of *message sending events* and the set $?\Sigma = \{?m \mid m \in \mathbb{M}^I\}$ of *message receiving events* (i. e., $\Sigma = !\Sigma \cup ?\Sigma$ and $!\Sigma \cap ?\Sigma = \emptyset$). We also distinguish a *terminating event final* $\notin \Sigma$ as a special event indicating a successful termination, and an *internal event* $\tau \notin \Sigma \cup \{final\}$ indicating an event that does not communicate with other services.

We define a *service automaton* as a non-deterministic finite state automaton equipped with two disjoint sets of input and output message channels, where each transition is labeled with either a communication event or an internal event.

Definition 2.1 (Service automaton).
A *service automaton* $S = [Q, q_0, I, O, \rightarrow, \Omega]$ consists of

- a finite set Q of states;
- an initial state $q_0 \in Q$;
- two disjoint sets of message channels; a set $I \subseteq \mathbb{M}^I$ of input message channels and a set $O \subseteq \mathbb{M}^O$ of output message channels;
- a non-deterministic transition relation $\rightarrow \subseteq Q \times \Sigma \cup \{\tau\} \times Q$ where $\Sigma = !\Sigma \cup ?\Sigma$; and
- a set $\Omega \subseteq Q$ of final states.

The interface of a service automaton is the union of the input and output message channels. The transition of a service automaton is labeled with either a communicating event $e \in \Sigma = !\Sigma \cup ?\Sigma$ or an internal event τ. An event *final* is a special event that represents the terminating event and therefore can be used to describe a final state $q \in \Omega$ of a service automaton. In contrast to the terminating event *final* and the internal event

τ, every communicating event can be realized together with other services. Though not communicating with other service automata, both a terminating event *final* and an internal event τ are *observable* from other service automata. Nevertheless, the detail of a τ event is irrelevant to other service automata.

As notational convention throughout the thesis, we write $q \xrightarrow{e} q'$ for a transition $[q, e, q'] \in \rightarrow$. If there exists a state q' such that $q \xrightarrow{e} q'$, we write $q \xrightarrow{e}$. We denote a service automaton S as *deterministic* whenever for each two transitions $q \xrightarrow{e} q'$ and $q \xrightarrow{e} q''$ of S it implies $q' = q''$. In case it is not clear from the context, we add an index to the constituents of a service automaton S, for instance, Q_S and $\rightarrow_S$.

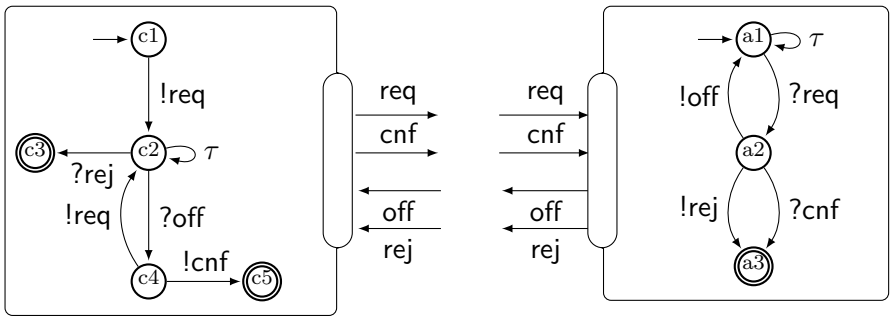
We write $q \xrightarrow{*} q'$, if there exists a (possibly empty) sequence $q \xrightarrow{e_1} \ldots \xrightarrow{e_n} q'$ of transitions from state q to state q', that is, state q' is *reachable from* state q. For the set $\Sigma = !\Sigma \cup ?\Sigma$ of communicating events, we denote the set of all *words* over Σ as Σ^* and denote an empty word by ϵ. For a communicating event $e \in \Sigma$ and a word $\sigma \in \Sigma^*$, the relation $\stackrel{\sigma}{\Rightarrow}$ is the least relation that satisfies the followings:

- $q \stackrel{\epsilon}{\Rightarrow} q$;
- $q \stackrel{\sigma}{\Rightarrow} q' \wedge q' \xrightarrow{e} q''$ implies $q \stackrel{\sigma e}{\Rightarrow} q''$; and
- $q \stackrel{\sigma}{\Rightarrow} q' \wedge q' \xrightarrow{\tau} q''$ implies $q \stackrel{\sigma}{\Rightarrow} q''$.

The relation $\Rightarrow$ only considers sequences of communicating events, whereas the relation $\rightarrow$ considers sequences of communicating events together with an internal event τ.

As we employ a service automaton to model the behavior of a service, we use the two notions *service* and *service automaton* interchangeably throughout the thesis.

Service automata for modeling and reasoning about service behavior has been introduced in Massuthe and Schmidt [2005]. Service automata model is closedly related to *Input/Output automata* [Lynch and Tuttle, 1989, Lynch, 1996], *Input/Output labeled transition system* [Tretmans, 1996], *interface automata* [de Alfaro and Henzinger, 2001], and *message exchange on finite state automata* [Baldoni et al., 2006]. We refer to Section 2.5 for a detailed comparison.



(a) Customer C (b) Agency A

Figure 2.1.: Two service automata; Customer service C and Agency Service A.

Example 2.2. Figure 2.1 illustrates two services; the customer service C and the travel agency service A, each is modeled with a service automaton. In the graphical representation of a service automaton, an arc without source state indicates the initial state, and double circles indicate the final states.

The customer service C can send a request (*req*) to an agency service and waits for either an offer (*off*) or a reject (*rej*). While waiting for a request message, it can perform (possible infinitely many) invisible events (τ). In case the customer service receives a reject message, the customer service terminates. Otherwise, it decides whether to send back either a confirm (*cnf*) and terminates or sends another request message and then waits for either an offer or a reject message from an agency service.

The agency service A initially is able to perform (possible infinitely many) invisible events while waiting for a request message. After receiving a request, the agency decides to send either an offer or a reject or waits to receive a confirm message. In case the customer service sends a reject or receives a confirm message, it then terminates. Otherwise, it sends an offer and then waits for another request message from a customer service. $\lhd$

A service is designed for communicating with other services, not for execution in isolation. We model the interplay between services with *service composition*, discussed in Section 2.2. In the remainder of this section, we distinguish between various states and components of a service automaton.

2.1.1. Stable State, Responsive State, and Divergent State

In this section, we define *stable state*, *responsive state*, and *divergent state* of a service automaton.

For each state q of a service automaton, we define the set of all events that are enabled at state q as follows.

Definition 2.3 (Enabled events, enabled communicating events).
For a state q of a service automaton S, we denote the set of all *enabled events* at state q by $\mathit{enable}(q) = \{e \mid q \xrightarrow{e} q'\} \cup \{\mathit{final} \mid q \in \Omega\}$.

For a state q of a service automaton, the set of all *enabled communicating events* of state q is defined by $\mathit{act}(q) = \mathit{enable}(q) \backslash \{\tau\}$.

For each state q of a service, $\mathit{enable}(q) \subseteq \Sigma \cup \{\tau, \mathit{final}\}$ whereas $\mathit{act}(q) \subseteq \Sigma \cup \{\mathit{final}\}$.

Next, we use the set of all enabled events at a given state to define *stable state* and *responsive state* as follows.

Definition 2.4 (Stable states, responsive states).
Let q be a state of a service automaton S. State q is a *stable* state of service automaton S iff $\tau \notin \mathit{enable}(q)$. State q is a *responsive* state of service automaton S iff $\mathit{enable}(q) \cap (I \cup O \cup \{\mathit{final}\}) \neq \emptyset$.

Intuitively, a stable state is a state which does not enable an internal τ event and a responsive state is a state which enables a communicating event in $I \cup O \cup \{\mathit{final}\}$.

The notion of stable states and responsive states can be used to describe a final state and a deadlock state of a service automaton. To this respect, a *final* state q is either a stable state or a responsive state that enables an event *final* (i. e., $\mathit{final} \in \mathit{enable}(q)$). A *deadlock* state q is a stable and non-responsive state q that does not enable any event including an internal event τ and a final event *final* (i. e., $\mathit{enable}(q) = \emptyset$). Clearly, a final state is always a responsive state whereas a deadlock state is never a responsive state.

Next, we distinguish a *divergent* state as a non-stable and non-responsive state that can perform nothing else but an infinite number of τ steps. We first define the set $\tau(q)$ of all *internally reachable* states of q of service S.

Definition 2.5 (Internally reachable states).
Let q be a state in service S. A state q' is *internally reachable* from q in service S iff state q' is reachable from state q by performing a (finite or infinite) number of invisible τ-event steps, i. e., $q \stackrel{\epsilon}{\Rightarrow} q'$.

The set of all *internally reachable states* of q is denoted by $\tau(q)$.

With the internally reachable states, we define a *divergent* state as follows.

Definition 2.6 (Divergent states).
A state q of service S is a *divergent* state iff for each state q' that is internally reachable from q holds: $\mathit{enable}(q') = \{\tau\}$, i. e., $\forall q' : q' \in \tau(q) :: \mathit{enable}(q') = \{\tau\}$.

Intuitively, a divergent state q in S is a non-stable and non-responsive state in S such that S *can diverge* from state q on. This means, from state q on, service S can only perform infinitely many τ steps such that after each τ step, none of the communicating events from $\Sigma \cup \{\mathit{final}\}$ is enabled.

Example 2.7. Figure 2.2 illustrates examples for three different types of states; stable state, responsive state, and divergent state.

We see that states q_1, q_2, and q_4 are stable states, states q_2, q_4, q_6, q_7, and q_8 are responsive states, whereas states q_9 and q_{10} are divergent states. Observe that states q_2 and q_4 are both stable and responsive states, whereas each of the states q_3 and q_5 is none of the three types. ◁

2.1.2. τ-Strongly Connected Components

For an automaton, a strongly connected component of an automaton is a fragment of an automaton in which there is a connected path from each state in the component to every other state in the same component.

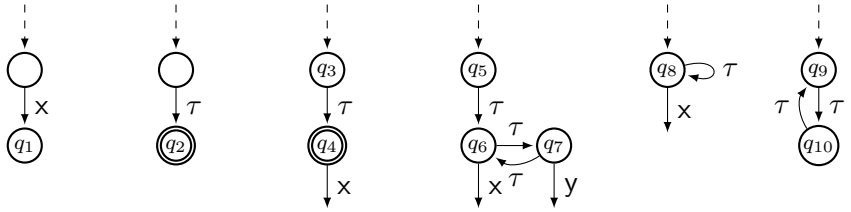


Figure 2.2.: Various types of states where q_1, q_2 and q_4 are stable states; q_2, q_4, q_6, q_7 and q_8 are responsive states; q_9 and q_{10} are divergent states.

In this section, we investigate a strongly connected component of a service automaton that has a distinguished property. With internally reachable states defined in the previous section, we first distinguish a τ-strongly connected component of a service automaton.

Definition 2.8 (τ-strongly connected component).
A *τ-strongly connected component* $\mathcal{C}$ of a service automaton S is a strongly connected component of S such that for each two states q and q' of $\mathcal{C}$ holds : q is internally reachable from q' and q' is internally reachable from q, i. e., $\forall q, q' \in Q_{\mathcal{C}} : q \stackrel{\varsigma}{\Rightarrow} q' \ \wedge \ q' \stackrel{\varsigma}{\Rightarrow} q$.

Next, we distinguish a τ-strongly connected component that is *stable* by checking whether every outgoing transition labeled with τ at every state within the component reaches a destination state within the same component.

Definition 2.9 (Stable τ-strongly connected component).
A τ-strongly connected component $\mathcal{C}$ of service automaton S is *stable* if for each state q of $\mathcal{C}$ with $q \stackrel{\tau}{\rightarrow}_S q'$ holds : q' is also a state in $Q_{\mathcal{C}}$, i. e., $\forall q \in Q_{\mathcal{C}} \ \wedge \ \exists q' : q \stackrel{\tau}{\rightarrow}_S q'$ it holds that $q' \in Q_{\mathcal{C}}$.

In the previous section, the set of all enabled communicating events of state q of a service automaton is defined by $act(q) = enable(q) \backslash \{\tau\}$. In the followings, we define the set of all enabled communicating events of a τ-strongly connected component C as the union of all enabled communicating events of all states within the component C.

Definition 2.10 (Enabled communicating events of component).
For a τ-strongly connected component of a service automaton, the set of all *enabled communicating events* of every state q in $\mathcal{C}$ is defined by $act^*(Q_{\mathcal{C}}) = \bigcup_{q \in Q_{\mathcal{C}}} act(q)$ where $Q_{\mathcal{C}}$ is the set of all states contained in $\mathcal{C}$.

Example 2.11. Figure 2.3 shows a fragment of a service automaton S with several τ-strongly connected components, each component is illustrated by a dashed rectangle.

There are two τ-strongly connected components that are not stable in Figure 2.3. These two are, the component that contains state q_1 and the component that contains state q_6. Each of the two components contains a state with an outgoing τ transition that connects to a state in another component. Other τ-strongly connected components shown in Figure 2.3 are stable.

In terms of its ability to communicate with the environment by performing an non-internal event, the component containing states q_4, q_5 and the component containing state q_7 are not distinguishable from its environment (i. e., other services). As they both enable the same set of communicating events; these are, events x and y. The same holds for the component containing state q_2 and the component containing state q_8, as they both do not enable any communicating event. ◁

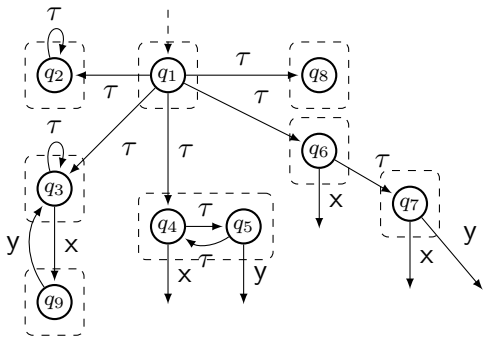


Figure 2.3.: A fragment of a service automaton with several τ-strongly connected components.

In the following lemma, we show that a stable τ-strongly connected component without an enabled communicating event contains either a deadlock state or a divergent state.

Lemma 2.12 (Divergent and deadlock states do not enable a communicating event).
For each stable τ-strongly connected component C of service automaton S holds : $act^*(Q_C) = \emptyset$ iff there exists a state $q \in Q_C$ such that q is a deadlock state or q is a divergent state.

Proof. We prove this lemma in two directions:

$\Rightarrow$: Suppose $act^*(Q_C) = \emptyset$; that is, $\bigcup_{q \in Q_C} act(q) = \emptyset$ by definition. Consider an arbitrary state $q \in Q_C$, this means, $act(q) = \emptyset$ must hold. As $act(q) = enable(q) \backslash \{\tau\}$ by definition, we conclude that q is either a deadlock state ($enable(q) = \emptyset$) or a divergent state ($\forall q' : q' \in \tau(q) :: enable(q') = \{\tau\}$).

$\Leftarrow$: Suppose q is a deadlock state in C; that is, $enable(q) = \emptyset$ by definition. This means, $act(q) = enable(q) \backslash \{\tau\} = \emptyset$ follows. As C is a stable τ-strongly connected

component by assumption and $enable(q) = \emptyset$, it follows that there is no other state $q' \in Q_C$ such that $q \neq q'$ by definition. Therefore, we conclude that $act^*(Q_C) = \emptyset$.

Suppose q is a divergent state in C; that is, $enable(q) = \{\tau\}$ by definition. This means, $act(q) = enable(q) \backslash \{\tau\} = \emptyset$ follows. As q is a divergent state, it follows that for each state q' with $q \stackrel{\epsilon}{\Rightarrow} q'$ holds: $enable(q') = \{\tau\}$. As C is a stable τ-strongly connected component by assumption, this means q' is also a state in Q_C by definition. Therefore, we conclude that $act^*(Q_C) = \emptyset$.

Thus, the lemma holds. □

2.2. Service Composition

A service is designed to communicate with other services, and not to execute in isolation. The interplay of two communicating services is realized by their *composition*. In this section, we formalize the composition of two service automata. As a preliminary requirement, the composition of two services S and P requires that S and P must be *composable*, that is, share neither their input message channels ($I_S \cap I_P = \emptyset$) nor their output message channels ($O_S \cap O_P = \emptyset$). This means, each message channel must be implemented by exactly one service automaton.

Definition 2.13 (Composable service automata).
Two service automata S and P are composable iff $I_S \cap I_P = \emptyset$ and $O_S \cap O_P = \emptyset$.

To model an asynchronous communication between two service automata, we assume an implicit *unordered* buffer of pending messages between service automata. If a message sending event is initiated by one service, then it is always possible for the service to do so by sending a message to the buffer. Naturally, a message receiving event is enabled by one service only when there is a respective message pending in the buffer. Nevertheless, each service can perform its message receiving event independent of the order of messages produced by their producer. This means, messages can be consumed from the buffer in a different order than the one they are produced. A successful termination of communication between both services is indicated by a state in which both reach their final states and the message buffer is empty.

We model the communication behavior of two service automata in terms of *composition*. Informally, the composition of two service automata is a service automaton whose states represent the cross product of the states from the two participating service automata and the states of the message buffer. The transition relation is driven by the states of the composition and the enabled events from each participating service.

Technically, we model a message buffer by the set of multisets over a set of message channels. For a set X of message channels with $X \subseteq \mathbb{M}$, we denote the set of all multisets over X with $Bags(X)$. We use list notion for multisets, for instance, we write $[a, a, b]$ for the multiset of $\{a \mapsto 2, b \mapsto 1, c \mapsto 0\}$ over $\{a, b, c\}$ and we denote the empty multiset by $[\,]$. The multiplicity $\mathcal{M}(m)$ of $m \in \mathcal{M}$ is the number of m elements in the multiset.

For $k \in \mathbb{N}$, we denote $Bags_k(X)$ as the set of multisets over X that is bounded by k (i. e., $\mathcal{M} \in Bags_k(X)$ implies $\mathcal{M}(m) \leq k$ for each $m \in \mathcal{M}$).

We define the composition of two service automata as follows.

Definition 2.14 (Composition of service automata).
The *composition* of two composable service automata S and P is the service automaton $S \oplus P = [Q, q_0, I, O, \rightarrow, \Omega]$ consisting of

- $Q = Q_S \times Bags(\mathbb{M}) \times Q_P$;
- $q_0 = [q_{0S}, [\,], q_{0P}]$;
- $I = (I_S \cup I_P) \backslash (I_S \cap O_P)$,
- $O = (O_S \cup O_P) \backslash (I_S \cap O_P)$,
- $\Omega = \Omega_S \times \{[\,]\} \times \Omega_P$, and
- $\rightarrow \subseteq\ Q \times \Sigma \cup \{\tau\} \times Q$ contains exactly the following elements:
  1. for all $m \in (I_S \cap O_P) \cup (O_S \cap I_P)$,
     - $[q_S, \mathcal{M}, q_P] \xrightarrow{\tau} [q'_S, \mathcal{M} + [m], q_P]$ iff $q_S \overset{!m}{\rightarrow}_S q'_S$;
     - $[q_S, \mathcal{M}, q_P] \xrightarrow{\tau} [q'_S, \mathcal{M} - [m], q_P]$ iff $q_S \overset{?m}{\rightarrow}_S q'_S$ and $m \in \mathcal{M}$;
     - $[q_S, \mathcal{M}, q_P] \xrightarrow{\tau} [q_S, \mathcal{M} + [m], q'_P]$ iff $q_P \overset{!m}{\rightarrow}_P q'_P$;
     - $[q_S, \mathcal{M}, q_P] \xrightarrow{\tau} [q_S, \mathcal{M} - [m], q'_P]$ iff $q_P \overset{?m}{\rightarrow}_P q'_P$ and $m \in \mathcal{M}$;
  2. for all $m \in I \cup O$:
     - $[q_S, \mathcal{M}, q_P] \xrightarrow{!m} [q'_S, \mathcal{M}, q_P]$ iff $q_S \overset{!m}{\rightarrow}_S q'_S$;
     - $[q_S, \mathcal{M}, q_P] \xrightarrow{?m} [q'_S, \mathcal{M}, q_P]$ iff $q_S \overset{?m}{\rightarrow}_S q'_S$;
     - $[q_S, \mathcal{M}, q_P] \xrightarrow{!m} [q_S, \mathcal{M}, q'_P]$ iff $q_P \overset{!m}{\rightarrow}_P q'_P$;
     - $[q_S, \mathcal{M}, q_P] \xrightarrow{?m} [q_S, \mathcal{M}, q'_P]$ iff $q_P \overset{?m}{\rightarrow}_P q'_P$;
  3. for $m = \tau$:
     - $[q_S, \mathcal{M}, q_P] \xrightarrow{\tau} [q'_S, \mathcal{M}, q_P]$ iff $q_S \overset{m}{\rightarrow}_S q'_S$,
     - $[q_S, \mathcal{M}, q_P] \xrightarrow{\tau} [q_S, \mathcal{M}, q'_P]$ iff $q_P \overset{m}{\rightarrow}_P q'_P$.

The composition of two interface composable services S and P yields a service $S \oplus P$ where all their compatible *open* channels ($I_S \cap O_P$ and $O_S \cap I_P$) are connected. Each pair of compatible channels yields a *closed* channel and cannot be used by any other service. The interface of $S \oplus P$ consists of the input and output channels of S and P that are not shared by S and P. A composition $S \oplus P$ is *closed* if every channel of $S \oplus P$ is closed and every event is internal, that is, the sets of input and output message channels of $S \oplus P$ are empty ($I_{S \oplus P} = O_{S \oplus P} = \emptyset$).

A state of the composed service $S \oplus P$ of S and P is a tuple $[q_S, \mathcal{M}, q_P]$ consisting of a state q_S of S, a multiset (i. e., bag) $\mathcal{M}$ of currently pending messages that were sent but not yet received, and a state q_P of P. The initial state $[q_{0S}, [\,], q_{0P}]$ consists of the two initial states q_{0S} of S and q_{0P} of P and the empty multiset $[\,]$ of the message buffer. An m-labeled sending transition adds in $S \oplus P$ one element m to the bag $\mathcal{M}$. Similarly, a transition receiving an m removes one m from $\mathcal{M}$. Internal transitions of S and P do not update $\mathcal{M}$. In the composition, all transitions that are associated with *closed* channels become internal transitions labeled with τ. Final states of the composed system are those where both services are in their respective final states, and the message bag is empty. In contrast to Lohmann [2010], we do not keep record of *closed* message channels as the *composition history* of a service. This means, we do not distinguish events that are associated with closed message channels.

Note, that it is also conventional to model the asynchronous communication by synchronous communication with explicitly introduced message buffers. We refer to Stahl [2009, Appendix A] and Mooij et al. [2010] for the relationship between our model and an synchronous communication model with explicit communication buffers.

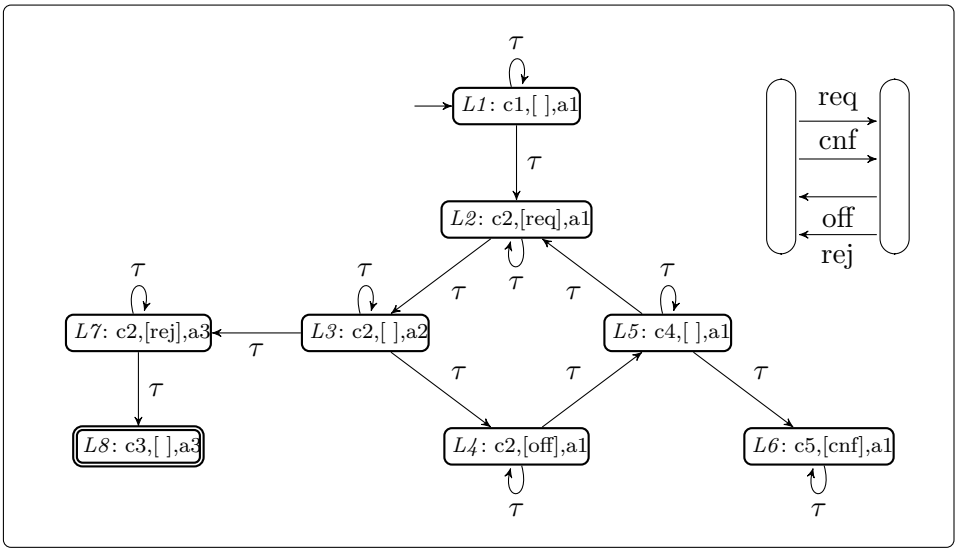


Figure 2.4.: The composition $C \oplus A$ of Customer C and Agency A.

Example 2.15. Figure 2.4 illustrates the composition $C \oplus A$ of Customer service C and Agency service A from Figure 2.1. Observe that the transition $a_2 \overset{?\text{cnf}}{\rightarrow}_A a_3$ of A are not covered in the composition $C \oplus A$. ◁

The order in which two services are composed is insignificant, though the cross product of two sets is in general not commutative. In our case, there is a bijective mapping between states $Q_{S\oplus P}$ of $S \oplus P$ and states $Q_{P\oplus S}$ of $P \oplus S$. As we assume that each interface must be implemented by exactly one service, the update of the multisets of messages is

always performed by the same service for each message production and consumption. By swapping the first and the third argument of states in the composition, $Q_{S \oplus P}$ and $Q_{P \oplus S}$ are isomorphic. Similar isomorphisms exist between the initial state and the set of final states of $S \oplus P$ and $P \oplus S$. Obviously, there is also a bijective mapping between input $I_{S \oplus P}$ and input $I_{P \oplus S}$ as well as between output $O_{S \oplus P}$ and output $O_{P \oplus S}$. By definition, the transition relation of the composed service is derived from the transition relations of both services interacting via the bags of messages. A sending event $!m$ or receiving event $?m$ of message m from one service becomes an internal event τ of the composition. An event of one service that is associated with an open channel of the composition remains the same in the composition, whereas an internal τ event of each service becomes also an internal τ event of the composition. Therefore, there exists a bijective mapping between the transition relation $\rightarrow_{S \oplus P}$ of $S \oplus P$ and the transition relation $\rightarrow_{P \oplus S}$ of $P \oplus S$ such that they are isomorphic.

To this respect, the composition operator is commutative and associative, which means the order of (pairwise) service composition is insignificant.

Proposition 2.16 (Composition is commutative and associative).
For each (pairwise) interface composable service automata S, P and R holds:

- the composition of service automata is commutative, i. e., $S \oplus P = P \oplus S$, and
- the composition of service automata is associative, i. e., $S \oplus (P \oplus R) = (S \oplus P) \oplus R$.

In this thesis, we study one fundamental aspect of correctness of services, that is, the *behavioral compatibility* of service composition. A compatibility criterion can be considered as a *Beauty* predicate $\mathcal{B}$ on service composition [Reisig, 2008] such that a service composition must satisfy the conditions posed by $\mathcal{B}$ in order to meet the criterion. For a given behavioral compatibility criterion $\mathcal{B}$, we denote the composition $S \oplus P$ that satisfies the criterion $\mathcal{B}$ by $\mathcal{B}(S \oplus P)$.

To reason about the correctness of service composition, we first require that every channel of service composition must be *closed* as it only makes sense to reason about service's behavior that is part of a *closed* composition. Informally, the closed composition of two service automata is the composition that has an empty set of interface (input and output message channels). To identify closed message channel of the composition, we define the interface compatibility of two service automata as follows.

Definition 2.17 (Compatible interface).
Two service automata S and P are *interface compatible* iff $I_S = O_P$ and $O_S = I_P$.

As for the next requirement, we must ensure that the composition $S \oplus P$ of services S and P has finitely many states by introducing the k-boundedness property of service composition, where $k \in \mathbb{N}$ is a message bound for each asynchronous message channel. For a message bound $k \in \mathbb{N}$, the composition of two services that satisfies the k-boundedness property is defined as follows.

Definition 2.18 (k-bounded service composition).
Let $k \in \mathbb{N}$ be a message bound for each message channel. The composition $S \oplus P$ is *k-bounded*, denoted by $k(S \oplus P)$, iff for each $[q_S, \mathcal{M}, q_P]$ of $S \oplus P$ and for each $m \in \mathbb{M}$ holds : $\mathcal{M}(m) \leq k$.

Clearly, if the composition $S \oplus P$ of two services S and P is *k-bounded* for $k \in \mathbb{N}$, then the composition $S \oplus P$ is also bounded by any number that is greater than k.

Example 2.19. In Figure 2.4, the composition $C \oplus A$ of Customer service C and Agency service A (from Fig. 2.1) is closed and bounded by k for $1 \leq k \leq \mathbb{N}$. The the interface of $C \oplus A$ is empty and each message channel does not store more than one message. $\lhd$

For the remainder of this section, we investigate *two behavioral compatibility criteria* $\mathcal{B}$ of service composition; k-deadlock freedom ($\mathcal{B} = df_k$) and k-responsiveness ($\mathcal{B} = rp_k$) for a message bound $k \in \mathbb{N}$ for each message channel. We discuss the k-deadlock freedom criterion in Section 2.2.1 and k-responsiveness criterion in Section 2.2.2.

2.2.1. Deadlock-free Composition

In this section, we investigate the k-deadlock freedom of service composition as defined by Lohmann et al. [2007a] and Massuthe [2009]. Informally, two services interact deadlock-freely whenever their composition does not contain a deadlock state; that is, every terminating sequence of events reachable in the composition must end with a final-state event.

With the service automata model, we can distinguish the desired terminal events which model a *successful completion* of service automaton from other terminal events such as a *deadlock* of service automaton. With the definition of a deadlock state introduced in Section 2.1.1, we define a *deadlock-free* service as a service automaton that does not contain a deadlock state.

Definition 2.20 (Deadlock-free service).
Service S is *deadlock-free* iff for each state q of S holds: q is not a deadlock state.

Note, that a deadlock state is a non-final state without an outgoing transition. It is possible that a deadlock-free service performs an infinite sequence of events that never terminates (known as *livelock*). Nevertheless the absence of deadlock in two service automata does not guarantee the absence of deadlock in their composition. On the contrary, a service automaton that contains a deadlock state may communicate with another and terminate successfully. As we model the composition by a service automaton, the composition of two service automata is deadlock-free whenever the service automaton that represents the composition of two service automata is also deadlock-free.

For a closed and k-bounded service composition for each message bound $k \in \mathbb{N}$, we define a k-deadlock-free service composition as follows.

Definition 2.21 (Deadlock-free service composition).
Let $k \in \mathbb{N}$ be a message bound for each message channel. Then, the composition $S \oplus P$ of two interface compatible service automata S and P is *k-deadlock-free*, denoted by $df_k(S \oplus P)$, iff

1. $S \oplus P$ is k-bounded; and
2. $S \oplus P$ is deadlock-free.

As a k-deadlock-free composition may contain an infinite sequence of events that never terminate (*livelock*), it is also possible that such a sequence includes only unbroken sequence of internal events (known as *divergence*) from one participating service. Naturally, a divergence within a composition excludes both a communicating event and a successfully terminating event from each participating service.

Example 2.22. The composition $C \oplus A$ illustrated in Figure 2.4 is k-deadlock-free for $k = 1$, as it does not contain a deadlock state; meaning, a non-final state without an outgoing transition. Observe state $L6$ of the composition. Though not a final state of the composition, $L6$ is also not a deadlock state of the composition. This is because there is an internal τ event enabled at $L6$ and it introduces an unbroken sequence of internal events in the composition. ◁

In this thesis, we also investigate a stronger behavioral compatibility criterion than k-deadlock freedom, called *k-responsiveness*. In a nutshell, a k-responsive composition is equivalent to a k-deadlock-free composition that is *divergence-free*. We discuss the k-responsive composition in the immediate following section.

2.2.2. Responsive Composition

In this section, we investigate the responsiveness of service composition as defined by Lohmann [2010]. Responsiveness is a stronger deadlock-freedom criterion than the one that is studied in Lohmann et al. [2007a] and Massuthe [2009], because it additionally requires the composition to exclude an unbroken infinite sequence of internal events in which each participating service can neither communicate with one another nor terminate successfully (known as *divergence*). In contrast to Lohmann [2010], we do not have the concept of *port*. Thereby, message channels can be grouped into ports from which interfaces are built. In this thesis, we consider only services with a single port in the sense of responsiveness defined by Lohmann [2010].

Similar to the basic requirements for deadlock-free composition, the responsive composition also requires that every channel of service composition must be *closed* as it only makes sense to reason about service's behavior that is part of a *closed* composition. This means, that they must have compatible interface. It also requires that the communication must not yield an unbounded number of pending messages in the buffer.

Informally, two services can interact responsively; whenever, (1) their composition is deadlock-free and (2) every sequence of events in their composition must not exclude

either a sending event, or a receiving event, or a successfully terminating event from each participating service in the composition. In this thesis, we formalize the condition for determining a responsive service composition differently from Lohmann [2010]. We determine the condition (2) by checking whether the *projecting behavior* of each participating service from a closed service composition is a *responsive* service.

With the definition of a stable τ-strongly connected component as introduced in Section 2.1.2, we define a *responsive service* as a service automaton in which each of its stable τ-strongly connected components enables a visible event.

Definition 2.23 (Responsive service).
Service S is *responsive* iff for each stable τ-strongly connected components $\mathcal{C}$ of S holds : $act^*(Q_{\mathcal{C}}) \neq \emptyset$.

Intuitively, a responsive service contains neither a deadlock state nor a divergent state. This is because every component is not a stable τ-strongly connected components in which it must contain either a deadlock state or a divergent state (cf. Lemma 2.12).

Given two responsive services, we cannot conclude that their composition is also responsive. One illustration for this phenomenon is when one service waits for consuming a message to be produced by the other, at the same time, it locally perform infinitely many sequence of invisible τ-events (known as a *local livelock*). In case that the other service fails to send the message, together they never perform any further communicating event. Intuitively, both services fail to *respond* to their own communicating partner.

Next, we define the projecting operator of service behavior from a closed service composition. As a closed service composition contains only invisible τ-events and possibly a final event, this operator produces the behavior of one participating service in the composition by renaming the respecting non-internal events of the other service participating in the composition to internal τ events.

Definition 2.24 (Respecting behavior of service composition).
The *behavior* of P with respect to the composition $S \oplus P = [Q, q_0, I, O, \rightarrow, \Omega]$ is a service automaton $Beh_{S \oplus P}(P) = [Q, q_0, I_B, O_B, \rightarrow_B, \Omega]$ where $I_B = I_P$, $O_B = O_P$, and $\rightarrow_B = Q \times (I_P \cup O_P \cup \{\tau\}) \times Q$ contains the following elements :

1. $[q_S, \mathcal{M}, q_P] \xrightarrow{\tau}_B [q'_S, \mathcal{M}, q_P]$ iff $[q_S, \mathcal{M}, q_P] \xrightarrow{\tau} [q'_S, \mathcal{M}, q_P]$,
2. $[q_S, \mathcal{M}, q_P] \xrightarrow{\tau}_B [q_S, \mathcal{M}, q'_P]$ iff $[q_S, \mathcal{M}, q_P] \xrightarrow{\tau} [q_S, \mathcal{M}, q'_P]$,
3. $[q_S, \mathcal{M}, q_P] \xrightarrow{\tau}_B [q'_S, \mathcal{M} + [m], q_P]$ iff $[q_S, \mathcal{M}, q_P] \xrightarrow{\tau} [q'_S, \mathcal{M} + [m], q_P]$ and $m \in O_S$;
4. $[q_S, \mathcal{M}, q_P] \xrightarrow{\tau}_B [q'_S, \mathcal{M} - [m], q_P]$ iff $[q_S, \mathcal{M}, q_P] \xrightarrow{\tau} [q'_S, \mathcal{M} - [m], q_P]$, $\mathcal{M}(m) > 0$, and $m \in I_S$;
5. $[q_S, \mathcal{M}, q_P] \xrightarrow{m}_B [q_S, \mathcal{M} + [m], q'_P]$ iff $[q_S, \mathcal{M}, q_P] \xrightarrow{\tau} [q_S, \mathcal{M} + [m], q'_P]$ and $m \in O_P$;
6. $[q_S, \mathcal{M}, q_P] \xrightarrow{m}_B [q_S, \mathcal{M} - [m], q'_P]$ iff $[q_S, \mathcal{M}, q_P] \xrightarrow{\tau} [q_S, \mathcal{M} - [m], q'_P]$, $\mathcal{M}(m) > 0$, and $m \in I_P$;

The respecting behavior of one service with respect to its service composition is a service automaton that represents the behavior of one service when interacting with another service in its service composition. To this respect, we can determine whether the service composition is responsive by checking whether the respecting behaviors of the participants are responsive.

We define the responsive service composition as follows.

Definition 2.25 (Responsive service composition).
Let $k \in \mathbb{N}$ be a message bound for each message channel. Then, the composition $S \oplus P$ of two interface compatible service automata S and P is *k-responsive*, denoted by $rp_k(S \oplus P)$, iff

1. $S \oplus P$ is k-bounded,
2. $Beh_{S \oplus P}(P)$ is responsive, and
3. $Beh_{S \oplus P}(S)$ is responsive.

Intuitively, a service composition is not k-responsive if either it is not k-bounded, or it contains deadlock state, or there exists one participating service that never responds to the other service in their composition. In the two latter cases, the projecting behavior of at least one participating service is not responsive and it contains a stable τ-strongly connected component with the empty set of enabled communicating events. The informal condition (2) of responsive composition has been reflected by condition 2. and 3. of the formal definition as both respecting behaviors of the composition are responsive and it guarantees that every sequence of events in the composition must not exclude either a sending communicating event, or a receiving communicating event, or a successfully terminating event from each participating service in the composition. Note, that responsiveness does not guarantee a termination of the composition, similarly to deadlock freedom.

Example 2.26. Figure 2.5 illustrates the respecting behavior $Beh_{C \oplus A}(C)$ of Customer service C (from Fig. 2.1) with respect to the composition $C \oplus A$ (from Fig. 2.4) of Customer service C and Agency service A (from Fig. 2.1). The behavior $Beh_{C \oplus A}(C)$ is not responsive as there is a stable τ-strongly connected component containing a state that does not enable a non-internal event from $I_C \cup O_C \cup \{\mathit{final}\}$, that is, a component containing state $L6$.

Figure 2.6 illustrates the respecting behavior $Beh_{C \oplus A}(A)$ of Customer service A with respect to the composition $C \oplus A$ of Customer service C and Agency service A. The behavior $Beh_{C \oplus A}(A)$ is not responsive as there is a stable τ-strongly connected component containing a state that does not enable a non-internal event from $I_A \cup O_A \cup \{\mathit{final}\}$, that is, the component containing state $L6$.

Therefore, we can conclude that the composition $C \oplus A$ is not responsive. ◁

The responsiveness does not require every sequence of events in a composition to terminate. This means, that responsiveness is a weaker criterion than *livelock freedom* [Wolf,

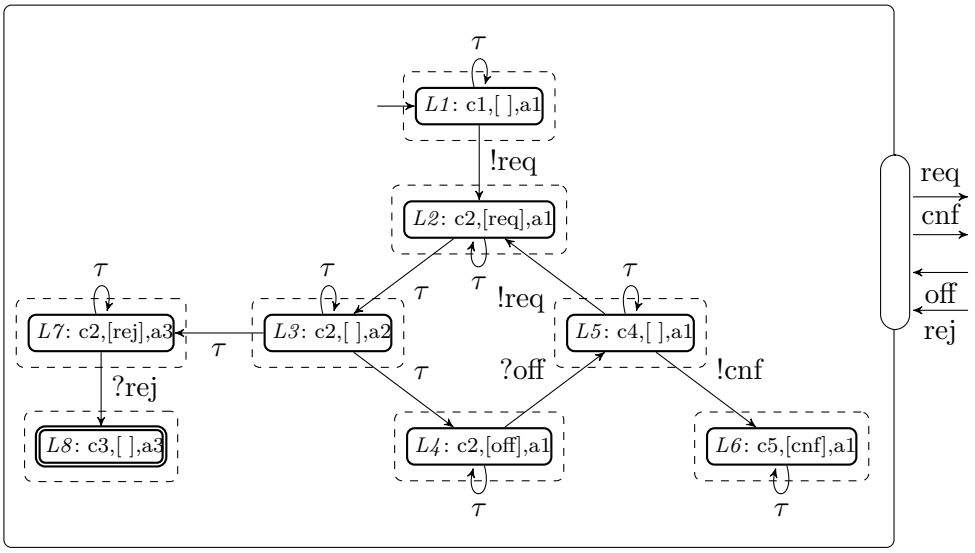


Figure 2.5.: The behavior $Beh_{C \oplus A}(C)$ of C with respect to the composition $C \oplus A$.

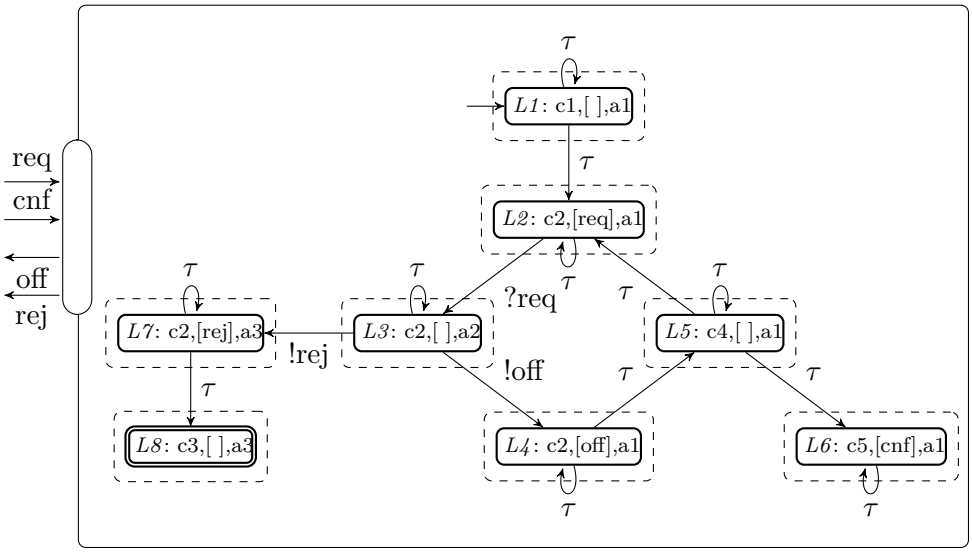


Figure 2.6.: The behavior $Beh_{C \oplus A}(A)$ of A with respect to the composition $C \oplus A$.

Stahl, Ott, and Danitz, 2009] which requires every sequence of events to terminate. The responsiveness criterion is also less strict than *weak termination* [Stahl, 2009, Massuthe et al., 2008] which requires every sequence of events to terminate successfully, and therefore, guarantees absence of deadlocks and livelocks. Weak termination coincides with the notion of *soundness* introduced for workflow nets by Van Hee, Sidorova, and Voorhoeve [2003]. In the branching-time computation tree logic (CTL) [Clarke and Emerson, 1982, Emerson and Halpern, 1985], weak termination can be expressed by $AG.EF final$ in which

it is always possible to reach a final state from every state of service composition. For an acyclic service, Stahl [2009] has shown that the notions of deadlock freedom and weak termination coincide. Similar coincidence holds for responsiveness, deadlock freedom, and weak termination.

2.3. Service and Controllers

In this section, we formalize the notion of a compatible partner of a given service and a given behavioral compatibility criterion $\mathcal{B}$ as a $\mathcal{B}$-*controller* of the given service. The notion *controller* originates from control theory of discrete event processes [Ramadge and Wonham, 1987, Cassandras and Lafortune, 2008] in the sense that we consider service P as a *controller* of service S for service S satisfying $\mathcal{B}(S \oplus P)$. The notion *controller* of a service is formalized in Lohmann et al. [2007a], Massuthe [2009], Stahl [2009] as *strategy* and in Lohmann [2010] as *partner*.

Two services S and P interact correctly with respect to a given behavioral compatibility criterion $\mathcal{B}$ whenever their composition $S \oplus P$ satisfies $\mathcal{B}$, that is, $\mathcal{B}(S \oplus P)$. In such case, service P is a $\mathcal{B}$-*controller* of service S. The notion of $\mathcal{B}$-controller is symmetric, as the composition operation $\oplus$ is commutative. This means, service P is a $\mathcal{B}$-controller of service S, whenever service S is also a $\mathcal{B}$-controller of service P.

For a service S with open interface, a closed composition that satisfies a given compatibility criterion $\mathcal{B}$ yields the concept of *controllability*. Service S is $\mathcal{B}$-*controllable* iff service S has at least one $\mathcal{B}$-controller, which means that there exists at least one service P with $\mathcal{B}(S \oplus P)$. The set of all $\mathcal{B}$-controllers (i. e., $\mathcal{B}$-compatible partners) of a service S is denoted by

$$\mathcal{B}\text{-}Controllers(S) = \{P \mid \mathcal{B}(S \oplus P)\}$$

where $\mathcal{B} \in \{df_k, rp_k\}$ represents either deadlock-freedom ($\mathcal{B} = df_k$) or responsiveness ($\mathcal{B} = rp_k$) of service composition for a message bound $k \in \mathbb{N}$.

Note, that the $\mathcal{B}$-controller notion is parameterized by a message bound $k \in \mathbb{N}$ on each message channel. As the k-bounded composition $S \oplus P$ of S and P implies that the composition $S \oplus P$ is also bounded by any number that is greater than k. For each compatibility criterion $\mathcal{B}$, service P is a $\mathcal{B}$-controller of service S for message bound $k \in \mathbb{N}$ implies service P is also a $\mathcal{B}$-controller of service S for a message bound that is greater than k. Though the reverse does not hold.

The following proposition states that the $\mathcal{B}$-*controller* relation between two services is symmetric, as the composition operator $\oplus$ is commutative.

Proposition 2.27 (Controller is a symmetric notion).
For each message bound $k \in \mathbb{N}$, each $\mathcal{B} \in \{df_k, rp_k\}$, and each two interface compatible service automata S and P with $\mathcal{B}(S \oplus P)$:

$$S \in \mathcal{B}\text{-}Controllers(P) \;\Leftrightarrow\; P \in \mathcal{B}\text{-}Controllers(S).$$

Next, we define the notions of k-deadlock-free controller and *k-deadlock-free controllability* for the k-deadlock-free compatibility criterion (i. e., $\mathcal{B} = df_k$). For two services S and P, their k-deadlock-free service composition yields the concept of *k-deadlock-free controller*.

Definition 2.28 (Deadlock-free controller).
Let $k \in \mathbb{N}$ be a message bound for each message channel. Then, service P is a *k-deadlock-free controller* of service S iff $df_k(S \oplus P)$. The set of all k-deadlock-free controllers of service S is denoted by

$$df_k Controllers(S) = \{P \mid df_k(S \oplus P)\}.$$

Service is *k-deadlock-freely controllable* iff service S has at least one k-deadlock-free controller, i. e., $df_k Controllers(S) \neq \emptyset$.

Example 2.29. The composition $C \oplus A$ illustrated in Figure 2.4 is a k-deadlock-free composition. This means, C and A are k-deadlock-free controller of each other, as well as both C and A are k-deadlock-freely controllable for $k = 1$. ◁

Next, we define the notions of k-responsive controller and *k-responsive controllability* for the k-responsive compatibility criterion (i. e., $\mathcal{B} = rp_k$). For two services S and P, their k-responsive service composition yields the concept of *k-responsive controller*.

Definition 2.30 (Responsive controller).
Let $k \in \mathbb{N}$ be a message bound for each message channel. Then, service P is a *k-responsive controller* of service S iff $rp_k(S \oplus P)$. The set of all k-responsive controllers of service S is denoted by

$$rp_k Controllers(S) = \{P \mid rp_k(S \oplus P)\}.$$

Service is *k-responsively controllable* iff service S has at least one k-responsive controller, i. e., $rp_k Controllers(S) \neq \emptyset$.

Example 2.31. As the composition $C \oplus A$ illustrated in Figure 2.4 is not k-responsive. Therefore, C and A are not k-responsive controller of each other. ◁

Given service S that is k-deadlock-freely controllable, it is not necessarily the case that service S is k-responsively controllable as illustrated by Property 2.32. Nevertheless, for a service S that is both k-deadlock-freely controllable and k-responsively controllable, the set of all k-responsive controllers of service S is a subset of the set of all k-deadlock-free controllers of service S.

Property 2.32 For each message bound $k \in \mathbb{N}$ and each service S that is k-deadlock-freely controllable and k-responsively controllable :

$$rp_k Controllers(S) \subseteq df_k Controllers(S).$$

2.4. Service Substitution Criteria

In this section, we formalize a notion of service substitution under a given behavioral compatibility criterion $\mathcal{B}$. In this thesis, we consider *controllers* defined in Section 2.3 as admissible contexts for service substitution.

We distinguish two different substitution criteria, each of which guarantees to preserve every controller of a given service before being substituted. Given a behavioral compatibility criterion $\mathcal{B}$, service T is a *substitute* for a service S *under* $\mathcal{B}$ *inclusion* if every $\mathcal{B}$-controller of service S is also a $\mathcal{B}$-controller of service T for a message bound k on each message channel. In case service S is also a *substitute* for service T *under* $\mathcal{B}$ inclusion, service S and service T have exactly the same set of $\mathcal{B}$-controllers and they can substitute one another under $\mathcal{B}$ *equivalence*.

As for basic requirements, substituting service S by service T first requires that the two services S and T must be *interface equivalent*; that is, they have the same set of input and output message channels ($I_S = I_T$ and $O_S = O_T$). As the behavior of a service (and its controller) is described by the specific order of external events which are determined by input and output message channels of a given service, this means, that reasoning about service substitutability only makes sense if the two services have the same set of communicating events.

Definition 2.33 (Equivalent interface).
Two service automata S and T are *interface equivalent* iff $I_S = I_T$ and $O_S = O_T$.

Substituting a service S by a service T also requires that the two services S and T must be $\mathcal{B}$-controllable according to a given behavioral compatibility criterion $\mathcal{B}$; this means $\mathcal{B}$-$Controllers(S) \neq \emptyset$ and $\mathcal{B}$-$Controllers(T) \neq \emptyset$. As $\mathcal{B}$-controllers are the context that must be preserved under substitution, it only makes sense to reason about correctness of service substitution if a service has at least one $\mathcal{B}$-controller.

In this thesis, we consider *two substitution criteria* for each behavioral compatibility criterion $\mathcal{B} \in \{df_k, rp_k\}$ representing either deadlock-freedom ($\mathcal{B} = df_k$) or responsiveness ($\mathcal{B} = rp_k$) of service composition.

Inclusion Substitution : Service T is a *substitute* for a service S *under* $\mathcal{B}$ *inclusion* iff every B-controller of service S is also a $\mathcal{B}$-controller of service T:

$$T \sqsubseteq_{\mathcal{B}} S \Leftrightarrow \mathcal{B}\text{-}Controllers(T) \supseteq \mathcal{B}\text{-}Controllers(S).$$

The $\sqsubseteq_{\mathcal{B}}$ is a *preorder* relation, as it is reflexive and transitive relation. Nevertheless, $\sqsubseteq_{\mathcal{B}}$ is not a partial order, as $T \sqsubseteq_{\mathcal{B}} S$ and $S \sqsubseteq_{\mathcal{B}} T$ does not implies that S and T are the same service.

The set of all substitutes for service S under $\mathcal{B}$ inclusion is denoted by

$$\mathcal{B}\text{-}\mathit{Inclusion}(S) \;=\; \{T \mid T \sqsubseteq_{\mathcal{B}} S\}.$$

The set $\mathcal{B}$-*Inclusion*(S) is never an empty set as service S is always a substitute for itself under $\mathcal{B}$-inclusion.

As $\mathcal{B}$-inclusion is a preorder relation, we use the two notions, $\mathcal{B}$-inclusion and $\mathcal{B}$-preorder, interchangeably throughout the thesis.

Equivalence Substitution : Service T is a *substitute* for a service S under $\mathcal{B}$ equivalence iff S and T have exactly the same set of $\mathcal{B}$-controllers:

$$T =_{\mathcal{B}} S \;\Leftrightarrow\; \mathcal{B}\text{-}\mathit{Controllers}(T) = \mathcal{B}\text{-}\mathit{Controllers}(S).$$

The relation $=_{\mathcal{B}}$ is an *equivalence* relation, as it is reflexive, transitive, and symmetric. This means, service T is a *substitute* for service S under $\mathcal{B}$ equivalence whenever service S is a *substitute* for service T under $\mathcal{B}$ equivalence.

The set of all substitutes for a service S under $\mathcal{B}$ equivalence is denoted by

$$\mathcal{B}\text{-}\mathit{Equiv}(S) \;=\; \{T \mid T =_{\mathcal{B}} S\}.$$

The set $\mathcal{B}$-*Equiv*(S) is never an empty set as service S is always a substitute for itself under $\mathcal{B}$ equivalence.

Note, that the notions of a substitute of a service under $\mathcal{B}$ inclusion and $\mathcal{B}$ equivalence are parameterized by a message bound $k \in \mathbb{N}$ on each message channel. As for each compatibility criterion $\mathcal{B}$, service P is a $\mathcal{B}$-controller of service S for message bound $k \in \mathbb{N}$ implies service P is also a $\mathcal{B}$-controller of service S for a message bound l that is that is greater than k ($k < l \leq \mathbb{N}$), though the reverse does not hold. Assume S is $\mathcal{B}$-controllable for both k and l. This means that a substitute for S under $\mathcal{B}$ inclusion for message bound k is not necessarily a substitute for S under $\mathcal{B}$ inclusion with a message bound l. On the contrary, a substitute for S under $\mathcal{B}$ inclusion for message bound l is also a substitute for S under $\mathcal{B}$ inclusion with a message bound k. We will study the relationship between a $\mathcal{B}$-controller of a given service and its substitute under $\mathcal{B}$ inclusion in Chapter 5.

Illustrated in Figure 2.7 are various sets of services related to a given service S. Each rectangle represents a set of services with equivalent interface (i. e., they have the same set of input and output message channels). Each dot represents a single service. These are the services S, T_1, T_2, and P. Each service in one rectangle has a compatible interface (i. e., an input message channel of one service is an output message channel of another and vice versa) with a service in the other rectangle. This means, the three services S, T_1, T_2 have equivalent interface, each of them has compatible interface with service P.

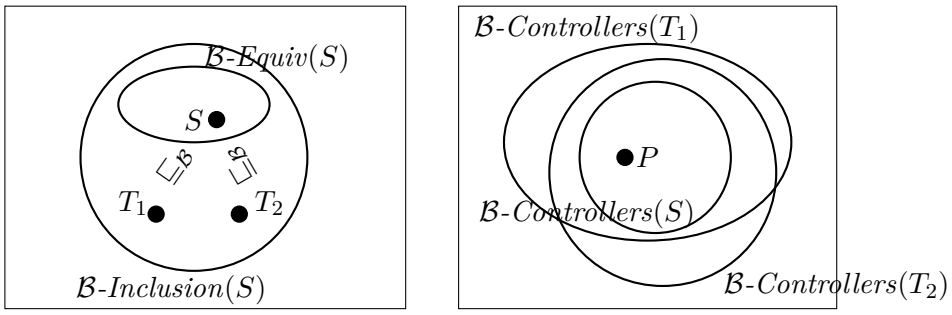


Figure 2.7.: Three interface equivalent services S, T_1, and T_2 where $T_1 \sqsubseteq_{\mathcal{B}} S$ and $T_2 \sqsubseteq_{\mathcal{B}} S$; each of the three has at least one common $\mathcal{B}$-controller P, for $\mathcal{B} = \{df_k, rp_k\}$ and $k \in \mathbb{N}^+$.

Each circle and each ellipse represents the set of services of particular interest with respect to service S. These sets are the $\mathcal{B}$-*Controllers*(S) of all controllers of service S, the $\mathcal{B}$-*Controllers*(T_1) of all $\mathcal{B}$-controllers of service T_1, the $\mathcal{B}$-*Controllers*(T_2) of all $\mathcal{B}$-controllers of service T_2, the $\mathcal{B}$-*Inclusion*(S) of all substitutes for service S under $\mathcal{B}$ inclusion, and the $\mathcal{B}$-*Equiv*(S) of all substitutes for service S under $\mathcal{B}$ equivalence. Figure 2.7 illustrates that both services T_1 and T_2 are substitutes for service S and service P is a $\mathcal{B}$-controller of each service S, T_1, and T_2.

With our convention for visualizing the set of services, the set represented by a circle or an ellipse never exceed beyond the boundary of a rectangle. This is because we assume that a service has a compatible interface with its controller, but same interface as its substitute service. We will use a similar convention to represent sets of services in every figure throughout the thesis.

In case $\mathcal{B} = df_k$, the notion of deadlock freedom inclusion is formalized by the notion of *accordance preorder* in Stahl et al. [2009], Parnjai et al. [2009], Mooij and Voorhoeve [2009], Mooij et al. [2011] and by *conformance preorder* in Stahl [2009]. In this thesis, we refer to these *preorder* as *deadlock freedom preorder* and to the respective and *equivalence* as *deadlock freedom equivalence*.

For the preorder relation $\sqsubseteq_{\mathcal{B}}$, it is important that the $\sqsubseteq_{\mathcal{B}}$ relation between two services must be preserved under the composition. This means, if service T is a substitute for service S under $\mathcal{B}$ preorder, then it is desirable that service $(T \oplus R)$ is also a substitute for service $(S \oplus R)$ under $\mathcal{B}$ preorder, for any service R that is composable with S and T. A preorder that satisfies this condition is a precongruence with respect to the composition operator $\oplus$. In the following lemma, we show a precongruence of $\sqsubseteq_{\mathcal{B}}$ with respect to the composition operator $\oplus$.

Lemma 2.34 ($\sqsubseteq_{\mathcal{B}}$ relation is a precongruence). [Stahl, 2009]
The $\sqsubseteq_{\mathcal{B}}$ relation is a precongruence with respect to the composition operator $\oplus$ for $\mathcal{B} \in \{df_k, rp_k\}$ and message bound $k \in \mathbb{N}$ for each message channel.

Proof. Let S and T be two interface equivalent services and R be a service that is composable with either S or T. Suppose $T \sqsubseteq_{\mathcal{B}} S$, by definition $\mathcal{B}$-*Controllers*$(T) \supseteq$ $\mathcal{B}$-*Controllers*(S) holds. We will show that $(T \oplus R) \sqsubseteq_{\mathcal{B}} (S \oplus R)$ also holds.

Suppose $P \in \mathcal{B}$-*Controllers*$(S \oplus R)$ holds. This means, $S \oplus R \oplus P$ is a service with empty set of input and output message channels such that $\mathcal{B}(S \oplus R \oplus P)$ holds. As $\oplus$ is commutative and associative (Proposition 2.16), $(R \oplus P) \in \mathcal{B}$-*Controllers*$(S)$ and $(S \oplus R) \in \mathcal{B}$-*Controllers*$(P)$ follows. As $T \sqsubseteq_{\mathcal{B}} S$ holds, this means T is a substitute for S under $\mathcal{B}$ inclusion. It follows that $(R \oplus P) \in \mathcal{B}$-*Controllers*$(T)$ holds by definition and $T \oplus R \oplus P$ is also a service with empty set of input and output message channels such that $\mathcal{B}(T \oplus R \oplus P)$ holds. It follows that $P \in \mathcal{B}$-*Controllers*$(T \oplus R)$ holds, because $\oplus$ is commutative and associative (Proposition 2.16).

Thus, we conclude that $(T \oplus R) \sqsubseteq_{\mathcal{B}} (S \oplus R)$ holds. □

For the relation $=_{\mathcal{B}}$, if service T and service S are equivalent under $\mathcal{B}$ equivalence, then it is desirable that service $(T \oplus R)$ is also a substitute for service $(S \oplus R)$ under $\mathcal{B}$ equivalence, for any service R that is composable with either S or T. In the following lemma, we show a congruence of $=_{\mathcal{B}}$ with respect to the composition operator $\oplus$.

Lemma 2.35 ($=_{\mathcal{B}}$ relation is congruence).
The $=_{\mathcal{B}}$ relation is a congruence with respect to the composition operator $\oplus$ for $\mathcal{B} \in \{df_k, rp_k\}$ and message bound $k \in \mathbb{N}$ for each message channel.

Proof. Follows from the definition of $=_{\mathcal{B}}$ and Lemma 2.34. □

2.4.1. Deadlock Freedom Inclusion

In this section, we define the notion of *deadlock freedom inclusion* for the deadlock freedom substitution criterion (i. e., $\mathcal{B} = df_k$).

Definition 2.36 (Deadlock freedom inclusion).
For a message bound $k \in \mathbb{N}$, service T can *substitute* service S under *k-deadlock freedom inclusion*, denoted by $T \sqsubseteq_{df,k} S$, iff df_k*Controllers*$(T) \supseteq df_k$*Controllers*(S). Then, the set of all substitutes for service S under k-deadlock-free inclusion is denoted by

$$df_k\textit{Inclusion}(S) \;=\; \{T \mid T \sqsubseteq_{df,k} S\}.$$

The notion *deadlock-free inclusion* has been formalized in Stahl [2009], Stahl et al. [2009], Parnjai et al. [2009], Mooij et al. [2011] as *accordance preorder*.

As deadlock-freedom inclusion is a preorder relation, we use the two notions, deadlock-freedom inclusion and deadlock-freedom preorder, interchangeably throughout the thesis.

2.4.2. Deadlock Freedom Equivalence

In this section, we define the notion of *deadlock freedom equivalence* for the deadlock freedom substitution criterion, i. e., $\mathcal{B} = df_k$.

The k-deadlock-free preorder relation $\sqsubseteq_{df,k}$ induces an equivalence relation $=_{df,k}$ that relates services with identical sets of k-deadlock-free controllers.

Definition 2.37 (Deadlock freedom equivalence).
For a message bound $k \in \mathbb{N}$, two services S and T are *equivalent under k-deadlock freedom* (or *k-deadlock-freely equivalent*), denoted by $S =_{df,k} T$, iff $S \sqsubseteq_{df,k} T$ and $T \sqsubseteq_{df,k} S$. Then, the set of all substitutes for service S under *k-deadlock-free equivalence* is denoted by

$$df_k Equiv(S) \;=\; \{T \mid T =_{df,k} S\}.$$

2.4.3. Responsiveness Inclusion

In this section, we define the notion of *responsiveness preorder* for the responsiveness substitution criterion (i. e., $\mathcal{B} = rp_k$).

Definition 2.38 (Responsiveness inclusion).
For a message bound $k \in \mathbb{N}$, service T is a *substitute* for service S under *k-responsiveness inclusion*, denoted by $T \sqsubseteq_{rp,k} S$, iff $rp_k Controllers(T) \supseteq rp_k Controllers(S)$. Then, the set of all substitutes for service S under k-responsive inclusion is denoted by

$$rp_k Inclusion(S) \;=\; \{T \mid T \sqsubseteq_{rp,k} S\}.$$

As responsiveness inclusion is a preorder relation, we use the two notions, responsiveness inclusion and responsiveness preorder, interchangeably throughout the thesis.

2.4.4. Responsiveness Equivalence

In this section, we define the notion of *responsiveness equivalence* for the responsiveness substitution criterion (i. e., $\mathcal{B} = rp_k$).

The k-responsiveness preorder relation $\sqsubseteq_{rp,k}$ induces an equivalence relation $=_{rp,k}$ that relates services with identical sets of k-responsive controllers.

Definition 2.39 (Responsiveness equivalence).
For a message bound $k \in \mathbb{N}$, two services S and T are *equivalent under k-responsiveness*, denoted by $S =_{rp,k} T$, iff $S \sqsubseteq_{rp,k} T$ and $T \sqsubseteq_{rp,k} S$. Then, the set of all substitutes for service S under *k-responsive equivalence* is denoted by

$$rp_k Equiv(S) \;=\; \{T \mid T =_{rp,k} S\}.$$

In the following example, we provide a set of interface equivalent service automata and illustrate the relation between each pair of service automata in the set.

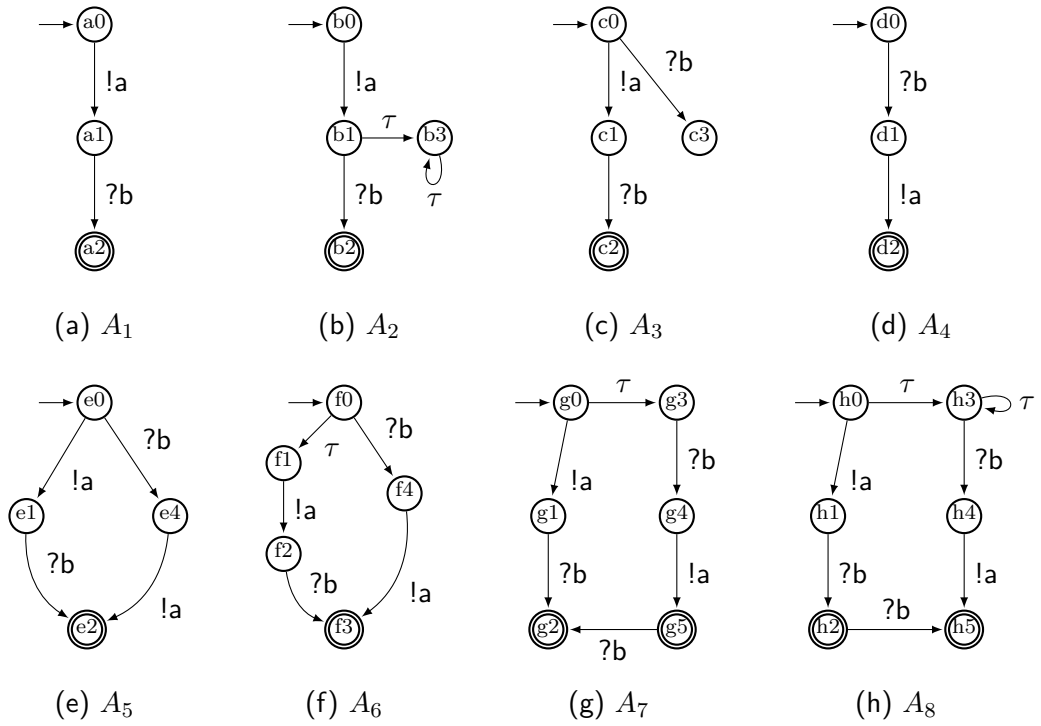


Figure 2.8.: Eight service automata: A_1, A_2, A_3, A_4, A_5, A_6, A_7, and A_8; each with interface $I = \{b\}$ and $O = \{a\}$.

Example 2.40. Figure 2.8 illustrates eight service automata with equivalent interfaces: A_1, A_2, A_3, A_4, A_5, A_6, A_7, and A_8. Each service automaton has the same set of input message channels ($I = \{b\}$) and the same output message channels ($O = \{a\}$) as the other services, though not all of them are in the same equivalence class according to either deadlock freedom or responsiveness. The illustration of the interface of each service automaton is omitted for readability purpose.

The following table illustrates the preorder ($\sqsubseteq_{\mathcal{B}}$) and equivalence ($=_{\mathcal{B}}$) relation between each pair of service automata with respect to both deadlock freedom ($\mathcal{B} = df_1$) and responsiveness ($B = rp_1$) for message bound $k = 1$ on each message channel.

In case $\mathcal{B}$ is presented as a label, this means, both deadlock freedom and responsiveness apply respectively to either preorder or equivalence.

In case either $\not{df}_1$ or $\not{rp}_1$ is given, this means, the service is respectively either not 1-deadlock-freely controllable or not is not 1-responsively controllable. For example, every service in Figure 2.8 is 1-deadlock-freely controllable and 1-responsively controllable, except for service A_2 which is 1-deadlock-freely controllable but not 1-responsively controllable.

	A_1	$A_2(\not{rp}_1)$	A_3	A_4	A_5	A_6	A_7	A_8
A_1	$=_{\mathcal{B}}$	$\sqsupseteq_{df,1}$	$\sqsubseteq_{\mathcal{B}}$	$\not\sqsubseteq_{\mathcal{B}}$ $\not\sqsupseteq_{\mathcal{B}}$	$=_{\mathcal{B}}$	$=_{\mathcal{B}}$	$\sqsubseteq_{\mathcal{B}}$	$=_{\mathcal{B}}$
$A_2(\not{rp}_1)$	$\sqsubseteq_{df,1}$	$=_{df,1}$	$\sqsubseteq_{df,1}$	$\not\sqsubseteq_{df,1}$ $\not\sqsupseteq_{df,1}$	$\sqsubseteq_{df,1}$	$\sqsubseteq_{df,1}$	$\sqsubseteq_{df,1}$	$\sqsubseteq_{df,1}$
A_3	$\sqsupseteq_{\mathcal{B}}$	$\sqsupseteq_{df,1}$	$=_{\mathcal{B}}$	$\not\sqsubseteq_{\mathcal{B}}$ $\not\sqsupseteq_{\mathcal{B}}$	$\sqsupseteq_{\mathcal{B}}$	$\sqsupseteq_{\mathcal{B}}$	$\not\sqsubseteq_{\mathcal{B}}$ $\not\sqsupseteq_{\mathcal{B}}$	$\sqsupseteq_{\mathcal{B}}$
A_4	$\not\sqsubseteq_{\mathcal{B}}$ $\not\sqsupseteq_{\mathcal{B}}$	$\not\sqsubseteq_{df,1}$ $\not\sqsupseteq_{df,1}$	$\not\sqsubseteq_{\mathcal{B}}$ $\not\sqsupseteq_{\mathcal{B}}$	$=_{\mathcal{B}}$	$\not\sqsubseteq_{\mathcal{B}}$ $\not\sqsupseteq_{\mathcal{B}}$	$\not\sqsubseteq_{\mathcal{B}}$ $\not\sqsupseteq_{\mathcal{B}}$	$\sqsubseteq_{\mathcal{B}}$	$\not\sqsubseteq_{\mathcal{B}}$ $\not\sqsupseteq_{\mathcal{B}}$
A_5	$=_{\mathcal{B}}$	$\sqsupseteq_{df,1}$	$\sqsubseteq_{\mathcal{B}}$	$\not\sqsubseteq_{\mathcal{B}}$ $\not\sqsupseteq_{\mathcal{B}}$	$=_{\mathcal{B}}$	$=_{\mathcal{B}}$	$\sqsubseteq_{\mathcal{B}}$	$=_{\mathcal{B}}$
A_6	$=_{\mathcal{B}}$	$\sqsupseteq_{df,1}$	$\sqsubseteq_{\mathcal{B}}$	$\not\sqsubseteq_{\mathcal{B}}$ $\not\sqsupseteq_{\mathcal{B}}$	$=_{\mathcal{B}}$	$=_{\mathcal{B}}$	$\sqsubseteq_{\mathcal{B}}$	$=_{\mathcal{B}}$
A_7	$\sqsupseteq_{\mathcal{B}}$	$\sqsupseteq_{df,1}$	$\not\sqsubseteq_{\mathcal{B}}$ $\not\sqsupseteq_{\mathcal{B}}$	$\sqsupseteq_{\mathcal{B}}$	$\sqsupseteq_{\mathcal{B}}$	$\sqsupseteq_{\mathcal{B}}$	$=_{\mathcal{B}}$	$\sqsupseteq_{df,1}$ $=_{rp,1}$
A_8	$=_{\mathcal{B}}$	$\sqsupseteq_{df,1}$	$\sqsubseteq_{\mathcal{B}}$	$\not\sqsubseteq_{\mathcal{B}}$ $\not\sqsupseteq_{\mathcal{B}}$	$=_{\mathcal{B}}$	$=_{\mathcal{B}}$	$\sqsubseteq_{df,1}$ $=_{rp,1}$	$=_{\mathcal{B}}$

From the table, the four services; A_1, A_5, A_6, and A_8 are in the same k-deadlock freedom equivalence class and responsive equivalence class for $k = 1$. Though when considering message bound $k \geq 2$ for each message channel, service A_8 is no longer in the same equivalence class as A_1, A_5, and A_6 due to the transition $h_2 \xrightarrow{?b} h_5$ in A_8 that can communicate with a controller that can send message b for the second time.

We see from the table that we cannot substitute service A_4 by any other service according to both k-deadlock freedom ($\mathcal{B} = df_1$) and k-responsiveness ($\mathcal{B} = rp_1$) inclusion. Nevertheless, service A_7 is a substitute for service A_4 under both deadlock freedom and responsiveness inclusion.

Services A_7 and A_8 illustrate the difference between deadlock freedom and responsiveness. For 1-responsiveness ($\mathcal{B} = rp, 1$), service A_7 is in the same equivalence class as services A_1, A_5, A_6, and A_8, for $k = 1$. For 1-deadlock-freedom ($\mathcal{B} = df, 1$), we can substitute service A_7 by either service A_1, A_5, A_6, or A_8 under 1-deadlock freedom, but not the other way around. ◁

2.5. Related Work

Service-Oriented Computing [Papazoglou, 2001, 2003] is an emerging paradigm that utilizes services as the basic construct to support the development of rapid, interoperable, evolvable, and massively distributed applications in heterogeneous environments. Recently, service-oriented computing has drawn substantial attentions from both industrial and academic domains. Peltz [2003] has illustrated the terms *choreography* and *orchestration* that are widely used to describe two aspects of composing services, though a strict separation is rather artificial [Papazoglou et al., 2007]. There are a number of service

description languages for specifying service orchestration such as WS-BPEL [Alves et al., 2007] and BPMN [OMG, 2009] as well as for specifying service choreography such as WS-CDL [Chinnici et al., 2003], Let's dance [Zaha et al., 2006], and BPEL4Chor [Decker et al., 2008]. Several research challenges of service-oriented computing have been addressed from various aspects [Papazoglou, 2003, Reisig et al., 2007, Papazoglou et al., 2007, 2008, Papazoglou, 2008b]. In this thesis, we address *service compatibility* and *service substitutability*, two important topics among the research challenges of service oriented computing [Papazoglou et al., 2008, Papazoglou, 2008b], using a fundamental approach to behavioral aspects of services [Reisig, 2008, Wolf, 2009b].

We study the observable behavior of *stateful* services [Papazoglou, 2008a] and model the control flow by describing the order in which asynchronous message are exchanged. Our description of service behavior is related to *communication protocol* [Merlin, 1979, Bochmann and Sunshine, 1980], *business interaction protocol* [Papazoglou, 2008a], and *conversation protocol* [Grigori et al., 2008]. Among all these notions, the communication protocol is the one that has been introduced before the service-oriented paradigm emerged, for the purpose of studying the order in which messages are exchanged and transfered.

Models that can distinguish choices made between either synchronous or asynchronous message passing are Petri nets [Reisig, 1985], finite state automata [Hopcroft et al., 2000], and process algebra [Bergstra et al., 2001, Baeten, 2005].

Petri nets have a successful history in the modeling, simulation, and verification of workflows and business processes. Along the line of Petri nets, *workflow nets* [van der Aalst, 1998] is a subclass of Petri nets for the modeling and verification of business processes. *Open nets* [Massuthe et al., 2005] is a subclass of Petri nets that has been tailored towards the modeling of services. There exists a bidirectional translation [Hinz et al., 2005, Lohmann and Kleine, 2008] between open net models and service description languages for creating service models such as WS-BPEL [Alves et al., 2007].

Automaton models have demonstrated the ability for modeling a large number of problems. Along the line of finite state automata, *Input/Output automata* [Lynch and Tuttle, 1989, Lynch, 1996] is a communication model that distinguishes explicitly between the inputs and the outputs of a system. Variants of the Input/Output automaton model are *Input/Output labeled transition system* [Tretmans, 1996, 1997], *interface automata* [de Alfaro and Henzinger, 2001], and *message exchange on finite state automata* [Baldoni et al., 2006]. These are synchronous communication models that provide supports for verification by making strong assumptions on the underlying infrastructure implementing the message exchange between the services. A comparative study of these models can be found in [Massuthe, 2009].

In this thesis, we employ *service automata* [Massuthe and Schmidt, 2005] as a uniform formalism to reason about correctness of services rather than to model services. The service automaton model is a variant of the Input/Output automaton model which performs asynchronous message passing via unidirectional message channels over an implicit unordered message buffer, allowing messages to overtake each other during their transmission. Stahl [2009] and Mooij, Stahl, and Voorhoeve [2010] have studied the relationship between asynchronous and synchronous communication models and illustrated how to model an asynchronous communication by synchronous communication with an explicitly

introduced message buffer. There exists a bidirectional translation between open Petri nets and service automata [Massuthe, 2009] and a variety of translations [Lohmann et al., 2009a,b] of service description languages into Petri nets and other formal models related to service automata.

The compatibility of services can be studied from various aspects [Papazoglou, 2008a]. At least four different aspects of service compatibility are described in the literature. *Syntactical compatibility* focuses on the service interface and ensures that message types between services must be compatible [Dong et al., 2004]. *Behavioral compatibility* guarantees absence of behavioral errors, such as deadlock and divergence [Bordeaux et al., 2005, Lohmann et al., 2007a, Dumas et al., 2008]. *Semantical compatibility* ensures correct interpretation of exchanging messages and their contents [Traverso and Pistore, 2004, Mrissa et al., 2007, Sycara et al., 2011]. *Non-functional compatibility* guarantees some quality parameters of services, such as security requirements and performance [Deora et al., 2006, Yu et al., 2007].

We study two behavioral compatibility criteria between services: *deadlock freedom* and *responsiveness*. Deadlock freedom has been studied in Lohmann et al. [2007a], Stahl et al. [2009], Massuthe [2009], Parnjai et al. [2009], Mooij and Voorhoeve [2009] and Mooij et al. [2011] for guaranteeing compatibility in the sense that the service composition will never get stuck in a deadlock state, a global non-final state in which none of participating services can perform any further action. Though one service may avoid a contribution to such a deadlock state by performing a local livelock [Wolf, 2009a, Wolf et al., 2009], a loop of internal actions. From this aspect, this behavior is related to *divergence* [Hoare, 1985b] from the area of process algebra where a process performs an unbroken sequence of internal events. Variants of deadlock freedom compatibility criterion have been studied in Müller [2010a,b] and Stahl and Vogler [2011]. Responsiveness has been formalized by Wolf [2009a] and Lohmann [2010]. It implies deadlock freedom and allows a service to perform a local livelock only if it is still able to a leave livelock state and respond to another service either by means of exchanging messages or be able to reach a final state whenever it is needed by another service in the composition. Though the termination of service composition is not guaranteed. In contrast to Lohmann [2010], we study responsiveness in the setting where a service has a single port [Wolf, 2009a]. A slight variant of responsiveness has been studied in Vogler et al. [2012]. Under the synchronous setting, responsiveness is closely related to *repetitive quiescence* [Tretmans, 1996] as it prevents one participating service to perform unbroken sequence of internal events.

To guarantee an absence of livelock, Wolf [2009a] and Wolf, Stahl, Ott, and Danitz [2009] have investigated *livelock freedom* of services, which guarantees that a service must terminate and proposed a technique for realizing a "find" request in a service-oriented architecture. Stahl [2009], Massuthe et al. [2008], Wolf et al. [2011], and Weinberg [2012] have studied the *weak termination* criterion which guarantees an absence of deadlocks and livelocks and therefore, it requires every sequence of events to terminate successfully. Weak termination coincides with the notion of *soundness* introduced for workflow nets [Van Hee, Sidorova, and Voorhoeve, 2003, van Hee, Mooij, Sidorova, and van der Werf, 2011]. In the computation tree logic (CTL) [Clarke and Emerson, 1982, Emerson and Halpern, 1985], weak termination can be expressed by $AG.EF\mathit{final}$ in which it is always possible

to reach a final state from every state of service composition. For an acyclic service that can not have a livelock, the notions of deadlock freedom, responsiveness, and weak termination coincide [Stahl, 2009, Wolf et al., 2009]. Stahl [2009] and Weinberg [2012] have studied the *strict termination* or *strong termination* criterion which guarantees that a final state must be indeed reached; therefore, a final state can never send, receive, or perform an internal action.

Controllability relates to the problem of deciding whether a given service can interact correctly with at least one partner [Wolf, 2009a]. The notion *controller* studied here is related to control theory of discrete event processes [Ramadge and Wonham, 1987, Cassandras and Lafortune, 2008]. The controller notion has been studied in various settings and formalized in Lohmann et al. [2007a], Wolf et al. [2009], Massuthe [2009], and Stahl [2009] as *strategy* and in Lohmann [2010] as *partner*. Weinberg [2008] has employed an interaction graph to analyze controllability of open nets. Gierds, Mooij, and Wolf [2010] have studied the *behavioral adapters* for adjusting incompatible communication between services and proposed an approach to synthesize an adapter by using existing controller synthesis algorithms. Wagner [2010, 2011], and Lohmann and Wolf [2011a] have investigated controllability of open nets in which data aspects are a major concern.

Annotated automata have been introduced by Wombacher et al. [2004] to represent sets of automata. Based on annotated service automata, *Operating Guidelines* [Massuthe et al., 2005, Massuthe and Schmidt, 2005, Lohmann et al., 2007a, Massuthe, 2009] is a technique for characterizing all controllers that satisfy deadlock freedom and responsiveness for asynchronously communicating services. A number of existing work is based on the operating guidelines approach, such as the characterization of controllers that enforce or exclude certain activities [Lohmann et al., 2007b], that cover a certain activity in less restrictive manner [Stahl and Wolf, 2008, 2009], and that must not exclude selected partners [Stahl et al., 2009]. Procedures for deciding service substitutability under various compatibility criterion have been proposed by Stahl [2009]. One of them is to decide service substitutability under deadlock freedom by means of comparing their operating guidelines [Stahl, Massuthe, and Bretschneider, 2009, Stahl and Wolf, 2009]. There are several efforts to optimize the space-efficiency of operating guidelines [Kaschner et al., 2007, Gierds, 2008a, Lohmann and Wolf, 2009]. A more compact representation of operating guidelines has been proposed by Lohmann and Wolf [2011b], where only few bits are stored within a state instead of a complete annotation, yielding improved efficiency in several procedures based on operating guidelines.

Service substitutability addresses one of the challenges of service oriented computing that is known as *business protocol change* [Papazoglou, 2008b]. The notion of deadlock freedom inclusion studied in this thesis has been formalized as *accordance preorder* in Stahl, Massuthe, and Bretschneider [2009], Parnjai, Stahl, and Wolf [2009], Mooij and Voorhoeve [2009] and Mooij, Parnjai, Stahl, and Voorhoeve [2011] and *conformance preorder* in Stahl [2009]. Substitutability is related to the problem of deciding whether the semantics of two given services satisfies the specific relation. In our context, service substitutability of two service is decided by comparing their sets of controllers and is related to the study of comparative concurrency semantics [van Glabbeek, 2001] in the area of process algebra.

Process algebra or process calculi [Bergstra et al., 2001, Baeten, 2005] provides a tool for formally modeling concurrent systems by representing interactions between independent processes as message-passing communication. In the field of process algebra research, there exists a number of process semantics known from the linear time-branching time spectrum [van Glabbeek, 1993, 2001], for instance, *traces* semantics [Hoare, 1978] as the coarsest one, *failures trace semantics* [Brookes et al., 1984, Hoare, 1978] as a finer one, and *bisimulation* semantics [Park, 1981, Milner, 1983] as the finest one. For expressing the same workflow in different workflow languages, Hidders et al. [2005] defines various equivalence notions in the presence of an internal τ event. It is shown that each equivalence does not coincide with classical notions of bisimulation under the presence of an internal τ event.

Along the line of process algebra, a number of related work to deadlock freedom and responsiveness preorders have been proposed to decide and characterize service substitutability in various settings. Stahl [2009] has shown that the *stable failures* preorder [Brookes et al., 1984, Hoare, 1978, 1985b, Roscoe, 1998] coincides with the deadlock freedom preorder on synchronous setting. Equivalence to stable failures for a finitely branching process without divergences is *must testing* [De Nicola, 1987]. Mooij et al. [2010] has considered related *fair testing* [Brinksma et al., 1995] as the coarsest preorder for weak termination, which implies weak termination preorder as studied in Stahl [2009] and Mooij et al. [2010]. Related to stable failure refinement is *Failure/divergence refinement* [Hoare, 1985b] for CSP which distinguishes processes before and after divergence. Though, Failure/divergence refinement does not coincide with our responsiveness inclusion as it considers a divergence process as a maximal element of the refinement preordered set of processes.

The following comparative study is based on the excellent survey by Stahl [2009]. Closest to the deadlock freedom inclusion is *subcontract preorder* [Laneve and Padovani, 2007] for CCS processes which coincides with τ-free services under deadlock freedom inclusion. A stronger notion of the subcontract preorder is *strong subcontract preorder* introduced by Bravetti and Zavattaro [2007] to guarantee absence of both deadlocks and livelocks, therefore, coinciding with weak termination conformance [Stahl, 2009, Mooij et al., 2010]. On the other hand, a more relaxed notion of the subcontract preorder is *weak subcontract preorder* introduced by Castagna et al. [2009] for filtering a set of actions of the service that may occur.

Stuck-free conformance [Fournet et al., 2004] is a refinement relation for CCS processes which guarantees the absence of deadlocks in an asynchronous communicating system, i. e., the interaction does not terminates with one partner getting stuck. Though Fournet et al. [2004] have shown that stuck-free conformance does not coincide with the stable failures preorder studied in Brookes et al. [1984]. The concept of responsiveness has been addresses with the notion *stable revivals* [Reed et al., 2007, Roscoe, 2009] for CSP models in response to stuck-free conformance for CCS models. Stable revivals requires a process to *respond to* a request of other process when expected. The study of stable revivals in Reed et al. [2007] is restricted to divergence-free processes, though our responsiveness allows local divergence in a service as long as it guarantee the absence of deadlock and divergence when services interact with each other.

Based on *pi calculus* [Milner et al., 1992], *session types* [Honda, 1993, Vasconcelos, 2009] have been introduced as a new form of polymorphism which allows typing of channel names by structured sequence of types. In contrast to our model, the order of actions specified by a session type must be followed strictly. Though the notion of a *canonical dual*, as originally introduced by Honda [1993], is comparable to a maximal controller in our setting. Related concept to duality is *Subtyping relation* [Vallecillo et al., 2003]. It allows a smaller session type to be safely used whenever a greater session type is expected.

Related to trace semantics, Stahl and Vogler [2011] has established a trace-based view on the operating guideline to analyze service substitutability under deadlock freedom preorder in a slightly different setting in which an interior of service does not have to be bounded when service is considered in isolation. This work also slightly modified the preorder relation (called *accordance*) in a more general refinement relation that is precongruence with respect to the composition operation. The work has been extended by Vogler et al. [2012] to responsiveness preorder. Though the characterization of all substitutes of a given service for both compatibility criteria are not provided. Based on prefix-closed trace structures, Dill [1989, 1990] has investigated *conformation preorder* and *conformation equivalence* to guarantee failure-free execution in a speed-independent hardware circuit. Dill [1989, 1990] has proposed to synthesize a *mirror*, as the maximum trace structure under the conformation ordering, and to check whether it can be composed with the original circuit component to yield a failure-free result in the system. The notion of mirror and the check conformation are respectively comparable to our canonical controller and its procedure to check inclusion, as studied in this thesis. A similar procedure has been employed to decide different preorders in different settings [Lin, de Jong, and Kolks, 1995a,b, Zhou, Yoneda, and Schlingloff, 2000].

Testing [Myers and Sandler, 2004, Tretmans, 1996] is a method which detects the presence of errors or flaws within a system. In this direction, [Kaschner, 2010, 2011] has investigated an approach to automatically generate test cases for asynchronously communication services using the notion of deadlock free conformance. Following the ideas of operating guidelines [Lohmann et al., 2007a, Kaschner and Wolf, 2009], the approach employs a test guideline for the purpose of generating test cases for testing a non-conformance partner.

The notion of a public view of a service has been addressed in van der Aalst and Weske [2001], Leymann et al. [2002], Eshuis and Grefen [2009], and van der Aalst et al. [2008]. In contrast to the construction algorithms in van der Aalst and Weske [2001] and Eshuis and Grefen [2009], our proposed public view based on a maximal controller is a canonical construction that is independent of reduction rules. Though, we propose a method to combine rule-based transformations with the construction of our public view. Our transformation rules are defined in the style of Murata rules Murata [1989] known for Petri nets. Some of the rules are inherited and extended from the literature [van der Aalst and Basten, 2002, van der Aalst et al., 2008, König et al., 2008, van der Aalst et al., 2009, Stahl, 2009].

Lohmann and Wolf [2010] have investigated how to implement a theory of correctness in the area of business processes and services and presents a family of tools, called

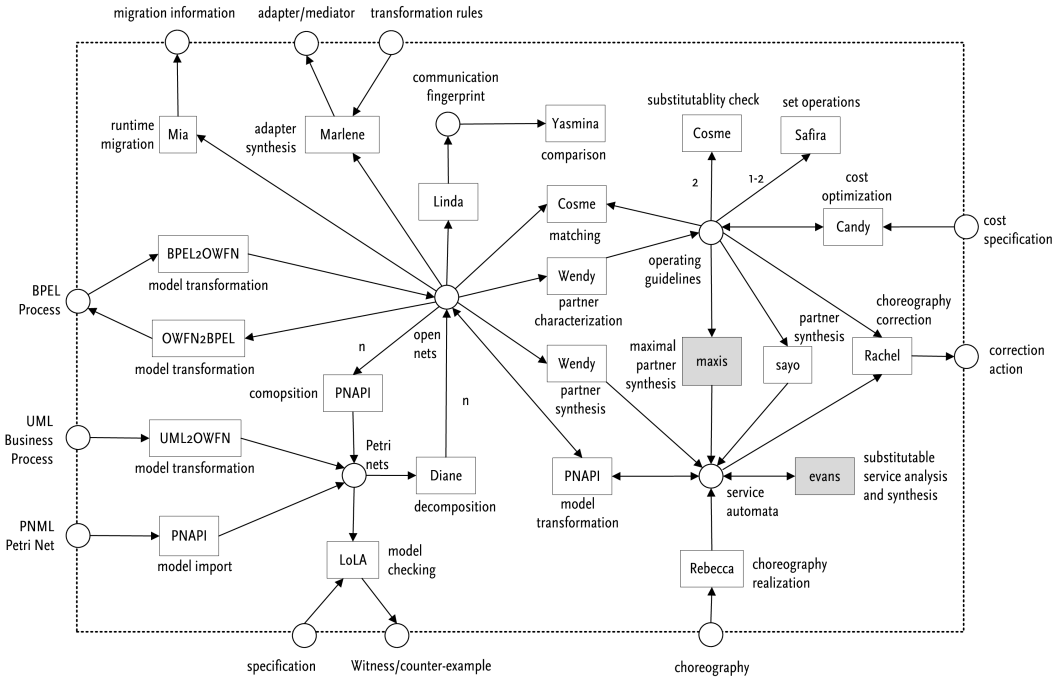


Figure 2.9.: Illustration of the tools family at *service-technology.org*

service-technology.org[1] (See Figure 2.9). The tool family provides support for a domain expert to explore our approach. For instance, the compiler *BPEL2oWFN* [Lohmann, 2007] and *PnAPI* [Mennicke et al., 2009] translates WS-BPEL processes into our formal model. Some tools provide support for various analysis tasks such as *Fiona* [Massuthe and Weinberg, 2008], *Wendy* [Lohmann and Weinberg, 2010], *Cosme* [Lehmann, 2011], *LoLA* [Schmidt, 2000, Wolf, 2007b], and *Rachel* [Lohmann, 2008b]. Two tools in the family, *Maxis* [Parnjai, 2011c] and *Evans* [Parnjai, 2011b] are developed under the course of this thesis. The complier *oWFN2BPEL* [Lohmann and Kleine, 2008] translates our formal model into an abstract *WS-BPEL* process. We will discuss about these tools in Chapter 7.

2.6. Concluding Remarks

In this chapter, we introduced the basic assumptions and notions that will be used in the subsequent chapters. We have employed *service automata* as a uniform formalism to model and to reason about correctness of service behaviors. Service automaton is a model with an explicit notion of states, in contrast to models with an implicit notion of states, such as Petri nets [Reisig, 1985] or process algebras [Bergstra et al., 2001, Baeten, 2005]. The behavioral compatibility criterion are characterized in terms of states

[1]available at `http://service-technology.org/tools`

and sequences of events of services and their composition. As we will introduce in the next chapter, an operating guideline technique [Massuthe et al., 2005, Massuthe and Schmidt, 2005, Lohmann et al., 2007a] for characterizing controllers of services are based on states. The algorithms to synthesize controllers presented in this thesis are developed on top of the operating guidelines approach and therefore, do not require an explicit express of concurrency. To create models of services, the domain-specific languages such as *WS-BPEL* [Alves et al., 2007], *BPMN* [OMG, 2009], and graphical formalisms such as Petri nets are more accessible to a domain expert. Nevertheless, there exists a bidirectional translation between service description languages such as into Petri nets and service automata models as well as to other formalisms related to automata [Lohmann et al., 2009a, Massuthe, 2009, Lohmann et al., 2009b].

3. Representing Sets of Services

This chapter introduces a technique for representing a set of service automata as introduced in the previous chapter. It illustrates how to employ this technique to characterize the sets of service automata with special properties, namely, the set of deadlock-free controllers and the set of responsive controllers with respect to a given service. The introduced techniques and notions will be investigated further in subsequent chapters.

The chapter is organized as follows. Section 3.1 introduces an annotated service automaton as a technique for representing the set of service automata. Section 3.2 discusses operations on annotated service automata. Section 3.3 presents two different finite representations of sets of services that satisfy the two behavioral compatibility criteria studied in this thesis. Section 3.4 concludes the chapter.

3.1. Annotated Service Automata

In this section, we present *annotated service automata* as a finite representation of the (possibly) infinite set of services. An annotated service automaton is a service automaton where each state is annotated with additional encoded information such as within a *Boolean formula*. Technically, a *matching relation* of the Boolean annotated service automaton characterizes all service automata in the set.

The concept of an *annotated automaton* has been introduced by Wombacher et al. [2004] to represent sets of automata. In [Lohmann et al., 2007a, Massuthe, 2009] and Stahl [2009] this concept has been employed to represent sets of deadlock-free controllers. Thereby, each Boolean formula is built from the set of events including the internal event τ and the successfully terminating event *final*, as well as the two Boolean operators; conjunction $\wedge$ and disjunction $\vee$. Lohmann [2010] has employed the concept of Boolean annotated service automata to represent sets of responsive controllers. Thereby, Lohmann [2010] has excluded an internal event τ from a Boolean formula and employed a different matching relation in comparison to the representation of deadlock-free controllers. In Kaschner and Wolf [2009], the Boolean formulae have been extended to include the negation Boolean operator $\neg$ for the purpose of realizing fundamental set operations of services and representing sets of responsive controllers.

In this thesis, we abstract from the encoded information within Boolean formulae. Instead, we annotate each state of an annotated automata with *choices*. In our context, a *choice* is a set of events representing a possible combination of events depending on a compatibility criterion. Intuitively, a choice is equivalent to one valid assignment to each Boolean annotated formula.

In this thesis, we study two behavioral compatibility criteria; deadlock freedom in the sense of Lohmann et al. [2007a], Massuthe [2009] and responsiveness in the sense of Lohmann [2010]. Due to technical reasons, we consider two different sets of events and two different matching relations corresponding to different compatibility criteria under investigation. We denote the set $\mathbb{E}$ of *action events* as follows.

– For deadlock freedom, we consider $\mathbb{E} = \Sigma \cup \{\tau\}$ as the set of action events.
– For responsiveness, we consider $\mathbb{E} = \Sigma$ as the set of action events.

For both cases, the set Σ is the set of communicating events that can be derived from the input and output message channels of a given service (see Section 2.1). We denote the set of all *action events* that include a successfully terminating event *final* by $\mathbb{E}_f = \mathbb{E} \cup \{final\}$.

The rest of this section is organized as follows. Section 3.1.1 distinguishes two different simulation relations. Section 3.1.2 introduces two different Boolean annotated service automata. Finally, Section 3.1.3 presents a choice service automaton as a generalization of a Boolean service automaton.

3.1.1. Simulation Relations

For comparing service automata, we distinguish two different simulation relations between service automata with respect to different compatibility criterion; these are the *strong*

simulation relation for deadlock freedom and the *structural simulation* relation for responsiveness.

We first define the *strong simulation* relation for service automata in the spirit of Milner [1971, 1989] defined for labeled transition systems. This definition is similar to the definition used in Lohmann et al. [2007a], Massuthe [2009] and Stahl [2009] for matching a deadlock-free controller.

Definition 3.1 (Strong simulation).
For two service automata T and U with equivalent interface, a binary relation $\varrho \subseteq Q_T \times Q_U$ between states of T and U is a *strong simulation relation* iff

1. $[q_{0T}, q_{0U}] \in \varrho$, and
2. for all states $q_T, q'_T \in Q_T$, all states $q_U \in Q_U$, and all $e \in \Sigma \cup \{\tau\}$ holds :
   if $[q_T, q_U] \in \varrho$ and $q_T \xrightarrow{e}_T q'_T$ then, there exists a state $q'_U \in Q_U$ with $q_U \xrightarrow{e} q'_U$ and $[q'_T, q'_U] \in \varrho$.

Then, U *strongly simulates* T if there exists a strong simulation relation ϱ of T by U.

We use a distinction between visible and invisible events to define another relation, *structural simulation* relation, for comparing behaviors of service automata in the spirit of Browne et al. [1987], as he defined it for labeled transition systems. This definition is similar to the definition employed by Lohmann [2010] for matching a responsive controller.

Definition 3.2 (Structural simulation).
For two service automata T and U with equivalent interface, a binary relation $\varrho \subseteq Q_T \times Q_U$ between states of T and U is a *structural simulation relation* iff

1. $[q_{0T}, q_{0U}] \in \varrho$ and
2. for all states $q_T, q'_T \in Q_T$, all states $q_U \in Q_U$, and all $e \in \Sigma \cup \{\tau\}$ holds :
   if $[q_T, q_U] \in \varrho$ and $q_T \xrightarrow{e} q'_T$, then either
   a) there exists a state $q'_U \in Q_U$ with $q_U \xrightarrow{e} q'_U$ and $[q'_T, q'_U] \in \varrho$, or
   b) $e = \tau$ and $[q'_T, q_U] \in \varrho$.

Then, U *structurally simulates* T if there exists a structural simulation relation ϱ of T by U.

3.1.2. Boolean Annotated Service Automata

A *Boolean annotated service automaton* is a service automaton where each state is annotated with a *Boolean formula*. The formulae are constructed on the *literals* that are elements of the set $\mathbb{E}_f$ of all action events including a terminating *final* event and *Boolean operators* that are the conjunctive Boolean operator $\wedge$, and the disjunctive Boolean operator $\vee$.

Definition 3.3 (Boolean formulae).
The set $\mathcal{BF}$ of *Boolean Formulae* over $\mathbb{E}_f$ is inductively defined as follows:

Basis: $e \in \mathbb{E}_f$ as well as *true* and *false* are Boolean formulae,

Step: if ϕ and ψ' are Boolean formulae, then $(\phi \vee \phi')$ and $(\phi \wedge \phi')$ are Boolean formulae.

The Boolean annotated service automaton is defined as follows

Definition 3.4 (Boolean annotated service automaton).
A *Boolean annotated service automaton* (or BSA for short) $U^{\phi} = [U, \phi]$ consists of

– a deterministic service automaton $U = [Q, q_0, I, O, \rightarrow, \Omega]$ and

– a mapping $\phi : Q \rightarrow \mathcal{BF}$, called *Boolean annotation.*

The truth value of an annotated formula is evaluated by an assignment function β. We distinguish two types of Boolean Assignment; *strong Boolean assignment* and *structural Boolean assignment.* The definition of *strong Boolean assignment* is similar to the definition used by Lohmann et al. [2007a], Massuthe [2009] and Stahl [2009] whereas the definition of *structural Boolean assignment* is similar to the definition used by Lohmann [2010].

Definition 3.5 (Strong boolean assignment).
A *Boolean assignment* (or strong assignment for short) β of a service automaton $S = [Q, q_0, I, O, \rightarrow, \Omega]$ is a mapping $\beta : Q \times \mathbb{E}_f \rightarrow \{true, false\}$ that is defined as follows:

$$\beta(q, e) = \begin{cases} true, & \text{if } e \in \mathbb{E} \text{ and there is a state } q' \text{ with } q \xrightarrow{e} q', \\ true, & \text{if } e = final \text{ and } q \in \Omega, \\ false, & \text{otherwise.} \end{cases}$$

A state q *strongly satisfies* a formula ϕ (denoted by $q \models_{\beta} \phi$) iff ϕ evaluates to *true* under the assignment β to each literal m using standard propositional logic semantics for the Boolean operators $\wedge$ and $\vee$.

Note, that we denote the set of all *action events* that include a successfully terminating event *final* by $\mathbb{E}_f = \mathbb{E} \cup \{final\}$. With respect to a strong Boolean assignment, an atomic proposition $e \in \mathbb{E}$ of a formula is true in a state q of a service automaton S if state q has a respective outgoing edge. The proposition *final* is evaluated to true if state q is in its final state. Otherwise, the proposition is evaluated to false.

Definition 3.6 (Structural boolean assignment).
A *structural Boolean assignment* (or structural assignment for short) γ of a service automaton $S = [Q, q_0, I, O, \rightarrow, \Omega]$ is a set mapping $\gamma : Q \times \mathbb{E}_f \rightarrow \{true, false\}$ that is defined as follows:

$$\gamma(q, m) = \begin{cases} true, & \text{if } m \in act^*(\tau(q)), \\ false, & \text{otherwise.} \end{cases}$$

A state q *structurally satisfies* a formula ϕ (denoted by $q \models_\gamma \phi$) iff ϕ evaluates to *true* under the assignment γ to each literal m using standard propositional logic semantics for the Boolean operators $\wedge$ and $\vee$.

With respect to a structural Boolean assignment, an atomic proposition $m \in \mathbb{E}$ of a formula is true in a state q of a service automaton S if either state q has a respective outgoing edge, possibly reached by a sequence of internal steps (cf. Definition 2.5 and Definition 2.10). The proposition *final* is evaluated to true if state q is final state.

In the followings, we define two different matching relations for a Boolean annotated service automaton that combines different simulation relations and Boolean Assignment functions. We first define the *Boolean strong matching relation* as follows.

Definition 3.7 (Boolean strong matching).
A service automaton T *strongly matches* with a Boolean annotated service automaton U^ϕ iff all the followings hold :

1. there exists a strong simulation relation $\varrho \subseteq Q_T \times Q_U$, and
2. for all $[q_T, q_U] \in \varrho$: q_T strongly satisfies $\phi(q_U)$, i. e., $q_T \models_\beta \phi(q_U)$.

The set of all service automata that strongly match with U^ϕ is denoted by $Match_\beta(U^\phi)$.

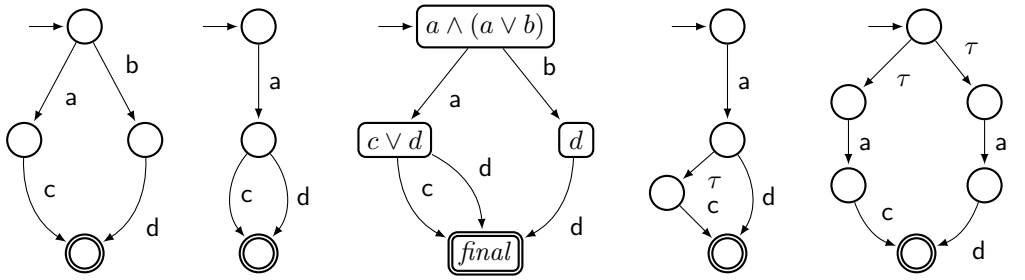
Next, we define the *Boolean structural matching relation* as follows.

Definition 3.8 (Boolean structural matching).
A service automaton T *structurally matches* with a Boolean annotated service automaton U^ϕ iff all the followings hold :

1. there exists a structural simulation relation $\varrho \subseteq Q_T \times Q_U$, and
2. for all $[q_T, q_U] \in \varrho$: q_T structurally satisfies $\phi(q_U)$, i. e., $q_T \models_\gamma \phi(q_U)$.

The set of all service automata that structurally match with U^ϕ is denoted by $Match_\gamma(U^\phi)$.

In the following example, we show a Boolean annotated service automaton and illustrate service automata that can be represented by the Boolean annotated service automaton using different matching relations.



(a) T_1 (b) T_2 (c) U^ϕ (d) T_3 (e) T_4

Figure 3.1.: A Boolean annotated service automaton U^ϕ and four service automata T_1, T_2, T_3, and T_4.

Example 3.9. Figure 3.1 illustrates one Boolean annotated service automaton U^ϕ and four service automata T_1, T_2, T_3, and T_4.

Each of the services T_1 and T_2 strongly matches with U^ϕ whereas both services T_3 and T_4 do not strongly match with U^ϕ due to their τ events and therefore they are not strongly simulated by U. This means that, with the strong matching relation, U^ϕ can represent T_1 and T_2, but not T_3 and T_4.

Nevertheless, each of the services T_1, T_2, T_3, and T_4 structurally matches with U^ϕ. This means that, with the structural matching relation, U^ϕ can represent T_1, T_2, T_3, and T_4. ◁

3.1.3. Choice Annotated Service Automata

In this section, we abstract from the Boolean formulae annotated at a given service automaton and encode all valid assignments of a *Boolean formula* into the structure of service automaton by using the concept of *choices*. Intuitively, each *choice* is the set of events that comprises of one valid assignment of the respective Boolean formula. In case a Boolean formula encodes *true*, any combination of events is allowed to include an empty set of events. In case a Boolean formula encodes *false*, only the empty set of events is allowed (see also Example 3.12).

We first define a *choice* as a set of events in $\mathbb{E}_f$.

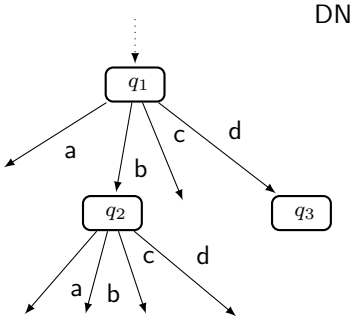
Definition 3.10 (Choice).
A *choice* ch at state q of a service automaton S is a (possibly empty) subset of the set $\mathbb{E}_f$ of all action events enabled at state q, i. e., $ch \subseteq \mathbb{E}_f$.

Intuitively, a choice represents a possible combination of various events that are allowed at state q of a given service automaton.

For each service automaton, we fix the set $\mathcal{CH}$ of all possible events for choices, i. e., $\mathcal{CH} = \mathbb{E}_f$. A service automaton which has each of its states annotated with a set of *choices* is called a *Choice annotated service automaton*.

Definition 3.11 (Choice annotated service automaton).
A *choice annotated service automaton* (or CSA for short) $Z^\alpha = [Z, \alpha]$ consists of a deterministic service automaton $Z = [Q, q_0, I, O, \rightarrow, \Omega]$ and $\alpha(q) \subseteq 2^{\mathcal{CH}}$, called *choice annotation* of state q, for each $q \in Q$.

Example 3.12. Figure 3.2 shows a comparison between sets of choices and the corresponding Boolean formulae. ◁



$$
\begin{array}{rcl}
\phi(q_1) & = & (a) \ \wedge \ (b) \ \wedge \ (a \vee b \vee c \vee d) \\
\text{DNF of } \phi(q_1) & = & (a \wedge b) \ \vee \ (a \wedge b \wedge c) \ \vee \ (a \wedge b \wedge d) \\
 & & \quad \vee \ (a \wedge b \wedge c \wedge d) \\
\alpha(q_1) & = & \{ \ \{a, b\}, \{a, b, c\}, \{a, b, d\}, \{a, b, c, d\} \ \} \\
 & & \\
\phi(q_2) & = & \mathit{true} \\
\alpha(q_2) & = & \{ \ \{ \ \}, \{a\}, \{b\}, \{c\}, \{d\}, \\
 & & \{ \ \{a, b\}, \{a, c\}, \{a, d\}, \{b, c\}, \{b, d\}, \\
 & & \quad \{a, b, c\}, \{a, b, d\}, \{a, b, c, d\} \ \} \\
 & & \\
\phi(q_3) & = & \mathit{false} \\
\alpha(q_3) & = & \{ \ \}
\end{array}
$$

Figure 3.2.: Comparison between Boolean formula $\phi(q)$ and the set $\alpha(q)$ of all choices at a state q.

The choice annotated service automaton is a generalization of the Boolean annotated service automaton. For given choices $\alpha(q)$ at state q, a complete disjunctive normal form (DNF) of a Boolean formula $\phi(q)$ of state q can be derived from $\alpha(q)$ as

$$\phi(q) = \bigvee_{\beta \in \alpha(q)} \bigwedge_{e \in \beta} e.$$

Similarly, choices $\alpha(q)$ at a given state q can be derived from a Boolean formula $\phi(q)$ at state q. Assume a Boolean formula at a given state q in its complete disjunctive normal form describing a sum of products of all events that are allowed at state q. This means that each summand describes a valid combination of events that can satisfy the assignment function of the Boolean formula at state q. As a choice represents a valid combination of events that are allowed at a given state q; therefore, a choice is equivalent to the set of events described by one summand, which is described by a Boolean formula $\phi(q)$ at state q.

In case a Boolean formula at a given state q is equivalent to *true*, any combination of events is valid, including an empty combination. This means, an empty set of events is also a choice at state q. On the contrary, in case a Boolean formula at a given state q is equivalent to *false*, every combination of events is invalid. Therefore, the set of all choices at state q is an empty set.

In order to decide whether a given service automaton is characterized by the choices annotated service automaton, we compare the structure of two service automata with a *simulation relation* and check whether a given service automaton *corresponds to* a choice at the corresponding state of the choices annotated service automaton.

Choice Correspondence

For checking whether a given service automaton *corresponds to* a choice annotated service automaton, we use two different *choice correspondence* relations with respect to different compatibility criteria; these are the *strong choice correspondence* relation for deadlock freedom and the *structural choice correspondence* relation for responsiveness.

Depending on a given choice correspondence relation, we refer to a *strong choice annotated service automaton* as a choice annotated service that uses a strong choice correspondence as matching relation and to a *structural choice annotated service automaton* as a choice annotated service automaton that uses a strong choice correspondence as matching relation. To this end, we define two different *choice correspondence* functions; *strong choice correspondence* and *structural choice correspondence* functions.

Definition 3.13 (Strong choice correspondence).
A strong correspondence β of the choice ch of a service automaton $S = [Q, q_0, I, O, \rightarrow, \Omega]$ is a mapping $\beta : Q \times 2^{\mathcal{CH}} \rightarrow \{true, false\}$ that is defined as follows:

$$\beta(q, ch) = \begin{cases} true, & \text{if } (\ ch = enable(q)\ ), \\ false, & \text{otherwise.} \end{cases}$$

A state q *strongly corresponds to* the choice ch iff $\beta(q, ch)$ evaluates to true.

For a state q of a service automaton, we denote the set of all communicating enabled events by $act(q) = enable(q) \backslash \{\tau\}$ (cf. Definition 2.3). For a set Q of states, we denote the set of all communicating enabled events by $act^*(Q) = \bigcup_{q \in Q} act(q)$ (cf. Definition 2.10).

Definition 3.14 (Structural choice correspondence).
A structural correspondence γ of the choice ch of a service automaton $S = [Q, q_0, I, O, \rightarrow, \Omega]$ is a mapping $\psi : Q \times 2^{\mathcal{CH}} \rightarrow \{true, false\}$ that is defined as follows:

$$\gamma(q, ch) = \begin{cases} true, & \text{if } (\ ch = act^*(\tau(q))\ ), \\ false, & \text{otherwise.} \end{cases}$$

A state q *structurally corresponds to* the choice ch iff $\gamma(q, ch)$ evaluates to true.

Compliance Matching

In the followings we define two different matching relations for a choice annotated service automaton; *strong compliance* and *structural compliance* relations, which combine different simulation relations and choice correspondence functions.

With the strong simulation relation and strong correspondence relation, we first define the strong compliance relation as follows.

Definition 3.15 (Strong compliance).
A service automaton T *strongly complies* with a choice annotated service automaton U^α iff

1. there exists a strong simulation relation $\varrho \subseteq Q_T \times Q_U$, and
2. for all $[q_T, q_U] \in \varrho$: there exists a choice $ch \in \alpha(q_U)$ such that q_T strongly corresponds to ch.

The set of all service automata that strongly comply with the choice annotated service automaton U^α is denoted by $Comply_\beta(U^\alpha)$.

The strong compliance relation between service T and a choice annotated service automaton U^α is decided by two conditions; first, if T is strongly simulated by U and second, if for each simulated pair $[q_T, q_U]$ of states q_T and q_U, state q_T strongly corresponds to ch.

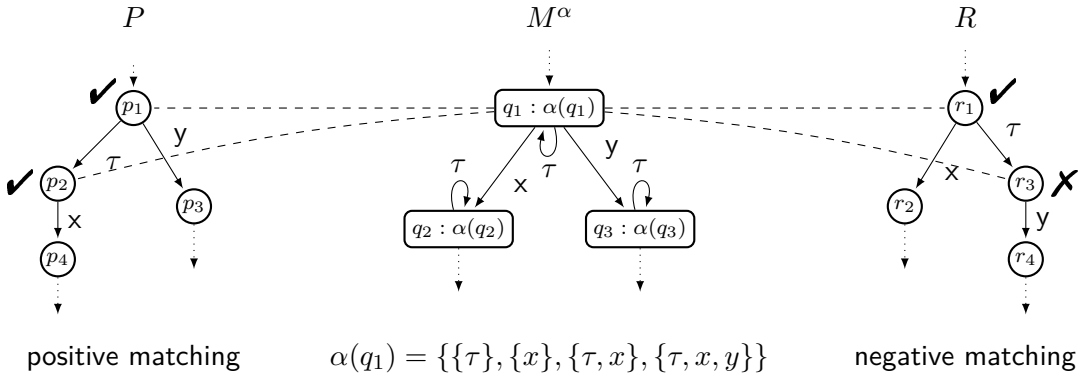


Figure 3.3.: Positive and negative matching of strong compliance of two services P and R with an annotated service automaton M^φ.

Example 3.16. Figure 3.3 shows a fragment of a choice annotated service automaton M^α with fragments of two service automata P and R; on the left hand side is a positive matching P and on the right hand is a negative matching R of M^α according to the strong compliance relation defined by Definition 3.15.

We assume both P and R are strongly simulated by M (strong simulation relation is denoted by dashed lines). Service R is a negative matching of M^α because at state r_3,

we have $enable(r_3) = \{y\}$ but there is no $ch \in \alpha(q_1)$ with $ch = \{y\}$). This means, for a pair of simulated states $[r_3, q_1] \in \varrho$ there exists no choice $ch \in \alpha(q_1)$ such that state $ch = enable(r_3)$; that is, r_3 does not strongly corresponds to ch. ◁

In Section 3.3.2, we will introduce the construction of one particular deadlock-free controller of a given service, called *most permissive deadlock-free controller* (defined in the sense of Lohmann et al. [2007a], Wolf [2009a], Massuthe [2009] and Stahl [2009]) and prove that the most permissive deadlock-free controller of a given service annotated by (strong) choice represents the set of all its deadlock-free controllers.

With the structural simulation relation and structural correspondence relation, we define the structural compliance relation as follows.

Definition 3.17 (Structural compliance).
A service automaton T *structurally complies* with a choice annotated service automaton U^α iff

1. there exists a structural simulation relation $\varrho \subseteq Q_T \times Q_U$, and
2. for all $[q_T, q_U] \in \varrho$: there exists a choice $ch \in \alpha(q_U)$ such that q_T structurally corresponds to ch.

The set of all service automata that structurally comply with U^α is denoted by $Comply_\gamma(U^\alpha)$.

The structural compliance relation between service T and a choice annotated service automaton U^α is decided by two conditions; first, if T is structurally simulated by U and second, if for each simulated pair $[q_T, q_U]$ of states the set of enabled events of state q_T is described by a choice annotated at state q_U of U.

Technically, the structural compliance delegates the check for matching τ event from structural choice correspondence to structural simulation relation. For applying structural compliance as a matching relation, we assume that each choice of U^α is built upon the set of action events that excludes an internal τ event.

Example 3.18. Figure 3.4 shows a fragment of a choice annotated service automaton M^α with fragments of two service automata P and R; on the left hand side is a positive matching P of M^α and on the right hand side is a negative matching R of M^α according to the structural compliance relation defined by Definition 3.17.

We assume both P and R are structurally simulated by M (structural simulation relation is denoted by dashed lines). Service P is a positive matching of M^α because we have $act^*(\tau(p_2)) = \{x, final\}$ and $act^*(\tau(p_1)) = act^*(\tau(p_3)) = \{x, y, final\}$. This means, for each pair of simulated states $[p, q]$ there exists a choice $ch \in \alpha(q)$ such that state p structurally corresponds to ch. Service R is a negative matching of M^α because at state r_3, we have $act^*(\tau(r_3)) = \{y\}$. However, there is no $ch \in \alpha(q_1)$ with $ch = \{y\}$. This means, for a pair of simulated states $[r_3, q_1] \in \varrho$ there exists no choice $ch \in \alpha(q_1)$ such that state r_3 structurally corresponds to ch. ◁

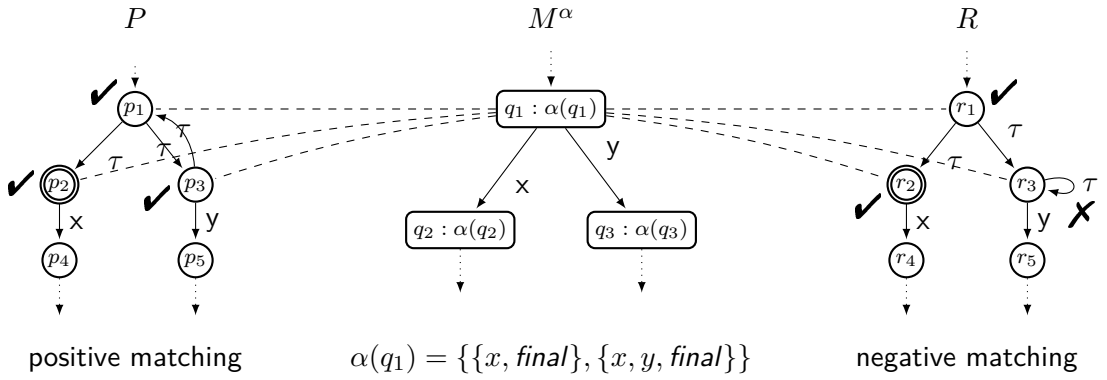


Figure 3.4.: Positive and negative matching of structural compliance of two services P and R with an annotated service automaton M^α.

For the responsiveness criterion, we will show in Section 3.3.3 the construction of one particular responsive controller of a given service, called *most permissive responsive controller* (defined in the sense of Wolf [2009a] and Lohmann [2010]) and prove that the most permissive responsive controller of a given service annotated by (structural) choice indeed represents the set of all its responsive controllers.

3.2. Operations on Annotated Service Automata

In the previous section, we introduced an annotated service automaton as a mean to represent of a set of service automata. In this section, we present a number of operations that can be performed upon an annotated service automaton.

The first operation, presented in Section 3.2.1, describes the intersection of two sets of service automata, each set is represented by an annotated service automaton. Section 3.2.2 presents a construction procedure of one distinguished service from a choice annotated service automaton. The constructed automaton is a member of the set described by the annotated service automaton which encodes within its own structure all possible sequences of events and all possible choices of every other service in the same set. Section 3.2.3 illustrates an alternative procedure that constructs a more compact representative of the set of services in comparison to the procedure introduced in Section 3.2.2.

3.2.1. Product of Annotated Service Automata

In this section, we present a product of two annotated service automata that characterizes the intersection of two sets of service automata, each of which is represented by an annotated service automaton. The intersection operator of two sets of services has been formalized in Bretschneider [2007], Stahl et al. [2009] and Stahl [2009] on Boolean annotated service automata. In this thesis, we generalize the intersection operator to choice annotated service automata.

The choice annotated service automaton S^α is a tuple $[S, \alpha]$ where S is a deterministic service automaton without final states, and α is an annotation at every state q of S. The product $S^\alpha \otimes R^\alpha$ of two choice annotated service automata S^α and R^α is also a choice annotated automaton that is an output of the product operation on the two inputs given as choice annotated service automata. A state of the product is a pair of two states from each automaton. The transition relation of the product is derived by those transitions that can be executed in both automata. An annotation at each state of the product is the intersection of the two sets of choices of the corresponding states.

Definition 3.19 (Product of choice annotated service automata).
Let $S^\alpha = [Q_S, q_{0S}, I_S, O_S, \rightarrow_S, \Omega_S, \alpha_S]$ and $R^\alpha = [Q_R, q_{0R}, I_R, O_R, \rightarrow_R, \Omega_R, \alpha_R]$ be two choice annotated service automata such that $(I_S \cap O_R) = (I_R \cap O_S) = \emptyset$. Then, the product choice state automaton $S^\alpha \otimes R^\alpha = [Q, q_0, I, O, \rightarrow, \Omega, \alpha]$ is defined by

- $Q = \varrho$ where $\varrho \subseteq Q_S \times Q_R$ is the strong simulation relation between states of S and R,
- $q_0 = [q_{0S}, q_{0R}]$,
- $I = I_S \cap I_R$,
- $O = O_S \cap O_R$,
- $[q_S, q_R] \xrightarrow{m} [q'_S, q'_R]$ iff $q_S \xrightarrow{m}_S q'_S$ and $q_R \xrightarrow{m}_R q'_R$, and
- $\alpha([q_S, q_R]) = \alpha(q_S) \cap \alpha(q_R)$, for all $[q_S, q_R] \in Q$.

Obviously, the product of two choice annotated service automata S^α and R^α can be an empty product. Intuitively, such a case means the empty intersection of the two sets of services represented by S^α and R^α.

The following corollary justifies that the product of two choice annotated service automata characterizes the intersection of the two sets of service automata, each represented by a choice annotated service automaton. The proof of the corollary has been generalized from the work of Stahl et al. [2009], Stahl [2009].

Corollary 3.20 (Intersection of two strong correspondence sets of services).
Let $T_\otimes = S^\alpha \otimes R^\alpha$ be the product of two choice annotated service automata S^α and R^α. Then $\mathit{Comply}_\beta(T_\otimes) = \mathit{Comply}_\beta(S^\alpha) \cap \mathit{Comply}_\beta(R^\alpha)$.

Proof. Let $S^\alpha = [Q_S, q_{0S}, I_S, O_S, \rightarrow_S, \Omega_S, \alpha_S]$, $R^\alpha = [Q_R, q_{0R}, I_R, O_R, \rightarrow_R, \Omega_R, \alpha_R]$, and $T_\otimes = [Q, q_0, I, O, \rightarrow, \Omega, \alpha_U]$. We prove this corollary in two directions.

$\Leftarrow$: Let $U \in \mathit{Comply}_\beta(T_\otimes)$ and ϱ_T be a strong simulation between U and $T_\otimes$. We show that $U \in \mathit{Comply}_\beta(S^\alpha)$ and $U \in \mathit{Comply}_\beta(R^\alpha)$ too.

Consider an arbitrary $[q_U, [q_S, q_R]] \in \varrho_T$. As ϱ_T is a strong simulation relation, there is a sequence $q_{0U} \stackrel{\sigma}{\Rightarrow}_U q_U$ such that q_U can be reached from q_{0U} via σ in U and $[q_S, q_R]$ can be reached from $[q_{0S}, q_{0R}]$ via σ in $T_\otimes$. The construction of $T_\otimes$

implies that q_S and q_R are reached via σ in S^α and R^α, too. By Definition 3.15, we have $[q_U, q_S] \in \varrho_S$ and $[q_U, q_R] \in \varrho_R$. Suppose there is an m-transition leaving q_U. From $U \in \mathit{Comply}_\beta(T_\otimes)$ and Definition 3.15, we can conclude that there is an m-transition leaving $[q_S, q_R]$, too. By the construction of $\rightarrow$ in Definition 3.19, there is an m-transition leaving q_S and one leaving q_R. Hence, q_S of S^α and q_R of R^α simulate q_U, too. Furthermore, we conclude from $U \in \mathit{Comply}_\beta(T_\otimes)$ and Definition 3.15 that q_U strongly corresponds to a choice $ch \in \alpha([q_S, q_R])$. Hence, by the construction of α in Definition 3.19, q_U also strongly corresponds to $\alpha(q_S)$ and $\alpha(q_R)$. Consequently, U strongly corresponds to S^α and to R^α and therefore $U \in \mathit{Comply}_\beta(S^\alpha) \cap \mathit{Comply}_\beta(R^\alpha)$ holds.

$\Rightarrow$: Let $U \in \mathit{Comply}_\beta(S^\alpha)$ and $U \in \mathit{Comply}_\beta(R^\alpha)$. We show that $U \in \mathit{Comply}_\beta(T_\otimes)$.

By $U \in \mathit{Comply}_\beta(S^\alpha)$ and Definition 3.15, there is a strong simulation relation ϱ_S between U and S^α.

Consider an arbitrary $[q_U, q_S] \in \varrho_S$. As ϱ_S is a strong simulation relation, there is a sequence $q_{0U} \stackrel{\sigma}{\Rightarrow}_U q_U$ and q_S is reached via σ in S^α. By $U \in \mathit{Comply}_\beta(R^\alpha)$ and Definition 3.15, there is a strong simulation relation ϱ_R and a state q_R such that $[q_R, q_U] \in \varrho_R$ and q_R is reached via σ in R^α. By the construction of $\rightarrow$ in Definition 3.19, $[q_S, q_R]$ is reachable in $T_\otimes$ via σ, too. Hence, $[q_S, q_R]$ of $T_\otimes$ simulates q_U, too.

Furthermore, we conclude from $U \in \mathit{Comply}_\beta(S^\alpha)$, $U \in \mathit{Comply}_\beta(R^\alpha)$, and Definition 3.15 that q_U strongly corresponds to a choice $ch_S \in \alpha(q_S)$ and q_U strongly corresponds to a choice $ch_R \in \alpha(q_R)$. Hence, by the construction of α in Definition 3.19, q_U also strongly corresponds to $\alpha([q_S, q_R])$, too. Consequently, U strongly corresponds to $T_\otimes$ and therefore $U \in \mathit{Comply}_\beta(T_\otimes)$ holds.

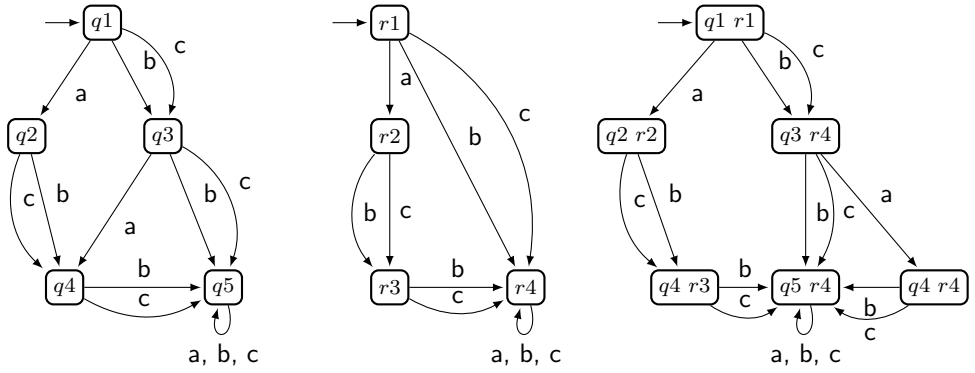
Thus, the corollary holds. □

Corollary 3.21 (Intersection of two structural correspondence sets of services).
Let $T_\otimes = S^\alpha \otimes R^\alpha$ be the product of two choice annotated service automata S^α and R^α. Then $\mathit{Comply}_\gamma(T_\otimes) = \mathit{Comply}_\gamma(S^\alpha) \cap \mathit{Comply}_\gamma(R^\alpha)$.

Proof. The proof of this corollary trivially follows from the proof of corollary 3.20, by considering transition that are internally reachable from state q instead of leaving state q directly as well as by replacing the strong simulation by the structural simulation and the strong correspondence by structural correspondence. □

Example 3.22. Figure 3.5 illustrates two services $OG_1 = [S_1, \alpha]$ and $OG_2 = [S_2, \alpha]$ and their product $OG_1 \otimes OG_2$.

In this example, OG_1 and OG_2 do not represent the same set of service automata with respect to both strong correspondence and structural correspondence. This is because



(a) OG_1 (b) OG_2 (c) $OG_1 \otimes OG_2$

$$
\begin{aligned}
\alpha(q1) &= \{\ \{a\},\{b,c\},\{a,\mathit{final}\},\{a,b\},\{a,c\},\\
&\quad \{a,b,c\},\{a,b,\mathit{final}\},\{b,c,\mathit{final}\},\\
&\quad \{a,b,c,\mathit{final}\}\ \}\\
\alpha(q2) &= \{\ \{b,c\},\{b,c,\mathit{final}\}\ \},\\
\alpha(q3) &= \{\ \{a\},\{a,\mathit{final}\},\{a,b\},\{a,c\},\\
&\quad \{a,b,c\},\{a,b,\mathit{final}\},\\
&\quad \{a,b,c,\mathit{final}\}\ \}\\
\alpha(q4) &= \{\ \{\mathit{final}\},\{b,\mathit{final}\},\{c,\mathit{final}\},\\
&\quad \{b,c,\mathit{final}\}\ \}\\
\alpha(q5) &= \{\ \{\ \},\{a\},\{b\},\{c\},\{\mathit{final}\},\\
&\quad \{a,\mathit{final}\},\{b,\mathit{final}\},\{c,\mathit{final}\},\\
&\quad \{a,b,c\},\{a,b,\mathit{final}\},\{a,c,\mathit{final}\},\\
&\quad \{b,c,\mathit{final}\},\{a,b,c,\mathit{final}\}\ \}
\end{aligned}
$$

$$
\begin{aligned}
\alpha(r1) &= \alpha(q3)\\
\alpha(r2) &= \{\ \{b,c,\mathit{final}\}\ \}\\
\alpha(r3) &= \alpha(q3)\\
\alpha(r4) &= \alpha(q5)
\end{aligned}
$$

$$
\begin{aligned}
\alpha(q1\ r1) &= \alpha(r1)\\
\alpha(q2\ r2) &= \alpha(r2)\\
\alpha(q3\ r4) &= \alpha(q3)\\
\alpha(q4\ r3) &= \alpha(q4)\\
&= \alpha(r3)\\
\alpha(q4\ r4) &= \alpha(q4)\\
\alpha(q5\ r4) &= \alpha(q5)\\
&= \alpha(r4)
\end{aligned}
$$

Figure 3.5.: The product $OG_1 \otimes OG_2$ of two choice annotated service automata OG_1 and OG_2.

there exists at least one service automaton that is represented by OG_1 but not by OG_2, that is a service automaton that can offer a choice $\{b, c\}$ after performing the first a. Also, there exists at least one service automaton that is represented by OG_2 but not by OG_1, that is a service automaton that can offer a choice $\{a\}$ after performing the a sequence c and a. This means that the set of service automata represented by OG_1 is not included in the set of service automata represented by OG_2 or vice versa.

We see that the product $OG_1 \otimes OG_2$ characterizes the intersection of the set of service automata represented by OG_1 and OG_2. ◁

We will investigate the product of annotated service automata in Chapter 7.

3.2.2. Construction of a Complete Representative of the Set

In this section, we present a construction procedure of one distinguished service from a choice annotated service automaton. The constructed service is a specific member

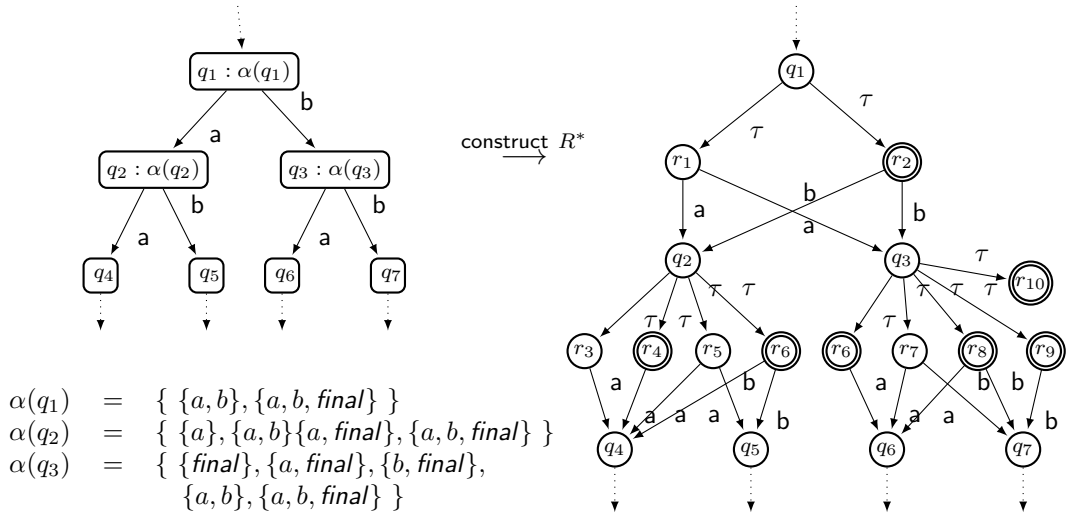
of the set described by the annotated service automaton that encodes within its own structure all possible sequences of events and all possible choices of every other service in the same set. This procedure will be used to construct a canonical controller discussed in Chapter 4, Section 4.1.

Given a choice annotated service automaton $R^\alpha = [R, \alpha]$, we construct one specific element R^* that is described by R^α from R^α by replacing each labeled state q of R with a fragment of nondeterministic internal τ events between all choices described at state q. The construction procedure of R^* from a choice annotated service automaton R^α is defined as follows.

Definition 3.23 (Construction of R^* from R^α).
Let $R = [Q, q_0, I, O, \rightarrow, \Omega]$ be a deterministic service automaton where each state $q \in Q$ is equipped with the set $\alpha(q)$ of all choices at q.

Then, the service $R^* = [Q^*, q_{0^*}, I, O, \rightarrow_*, \Omega^*]$ is defined as

- $Q^* = Q \ \cup \{q_{ch} \mid q \in Q \ \wedge \ ch \in \alpha(q)\}$
- $q_{0^*} = q_0$;
- $\Omega^* = \{q_{ch} \mid \mathit{final} \in ch \in \alpha(q)\}$;
- $\rightarrow^* = \{q \xrightarrow{\tau}_* q_{ch}\} \ \cup \{q_{ch} \xrightarrow{m}_* q' \mid q \xrightarrow{m} q' \ \wedge \ m \in ch \in \alpha(q)\}$.



$$
\begin{array}{lcl}
\alpha(q_1) & = & \{\ \{a,b\}, \{a,b,\mathit{final}\}\ \} \\
\alpha(q_2) & = & \{\ \{a\}, \{a,b\}\{a,\mathit{final}\}, \{a,b,\mathit{final}\}\ \} \\
\alpha(q_3) & = & \{\ \{\mathit{final}\}, \{a,\mathit{final}\}, \{b,\mathit{final}\}, \\
 & & \ \ \{a,b\}, \{a,b,\mathit{final}\}\ \}
\end{array}
$$

Figure 3.6.: Construction of R^* from a choice service automaton.

The construction procedure described by Definition 3.23 synthesizes from a choice annotated service automaton R^α a service R^* by replacing a state q with non-deterministic internal τ events of all choices described by $\alpha(q)$. By construction, service R^* and the

underlying service of the given annotated service automaton R have the same interface, but the constructed R^* is much larger than T in size. Obviously, the size of R^* is growing exponentially with respect to increased size of the service interface, which determines the set of all possible events of a service.

Example 3.24. Figure 3.6 shows a fragment of service R^* on the right hand side which is constructed from a fragment of a choice annotated service automaton R^α on the left hand side by applying Definition 3.23. ◁

In case of a strong choice annotated automaton R^α (where α is built on the set of actions including internal τ event and R is a deterministic service automaton with τ loop at every state q), the following lemma shows that R^* constructed from R^α is described by R^α with respect to the strong compliance relation.

Lemma 3.25 (Strong correspondence of R^*).
For each R^* that is constructed from a strong choice annotated automaton R^α :

$$R^* \in \mathit{Comply}_\beta(R^\varphi).$$

Proof. Let R^* be constructed from a strong choice annotated automaton $[R, \alpha]$ with $R = [Q, q_0, I, O, \rightarrow, \Omega]$ as described in Definition 3.23.

Let ϱ be a binary relation between states of R^* and of R with $\varrho = \{[q^*, q] \mid q \in Q \wedge \exists \beta \in \alpha(q) :: \beta = \mathit{enable}(q)\} \cup \{[q_\beta, q] \mid q \in Q \wedge \beta \in \alpha(q)\}$. As every state q in R has a τ-loop, and $\{\tau\} \in \alpha(q)$ holds, it follows that ϱ is a strong simulation relation and for each $[q^*, q] \in \varrho$ there exists $\beta \in \varrho(q)$ such that q^* strongly corresponds to β. Thus, $R^* \in \mathit{Comply}_\beta(R^\alpha)$ holds. □

In case of a structural choice annotated automaton R^α (where α is built on the set of actions excluding the internal τ event and R is a deterministic service automaton without τ loop at every state q), the following lemma shows that R^* constructed from R^α is described by R^α with respect to the structural compliance relation.

Lemma 3.26 (Structural correspondence of R^*).
For each R^* that is constructed from a structural choice annotated automaton R^α :

$$R^* \in \mathit{Comply}_\gamma(R^\alpha).$$

Proof. Let R^* be constructed from a structural choice annotated automaton $[R, \alpha]$ with $R = [Q, q_0, I, O, \rightarrow, \Omega]$ as described in Definition 3.23. Let ϱ be a binary relation between states of R^* and of R with $\varrho = \{[q^*, q] \mid q \in Q \wedge \exists \gamma \in \alpha(q) :: \gamma = \mathit{act}^*(\tau(q^*))\} \cup \{[q_\gamma, q] \mid q \in Q \wedge \gamma \in \alpha(q)\}$. As R is a deterministic and τ-free service automaton, it follows that ϱ is a structural simulation relation and for each $[q^*, q] \in \varrho$ there exists $\gamma \in \varrho(q)$ such that q^* structurally corresponds to β. Thus, $R^* \in \mathit{Comply}_\gamma(R^\alpha)$ holds. □

As R^* encodes within its own structure all possible sequences of events and all possible choices of every other service in the same set by construction, the constructed service R^* can be fairly large in size. In the next section, we present an alternative procedure for constructing a more compact representative of the set of services described by a given annotated service automaton.

We employ the construction of a complete representative of the set of services to synthesize a canonical controller and present some experimental results in Chapter 4.

3.2.3. Construction of a Compact Representative of the Set

In this section, we present an alternative procedure for constructing a more compact representative of the set of services in comparison to the procedure described in Section 3.2.2. Though not every service automaton in the set offers the same set of choices, there exists some choices that must be provided by every service automaton represented by the choice annotated service automaton. This procedure first computes a subset of minimal choices where each choice cannot be further reduced by eliminating from it any single event that is contained in other choices and regards it as a *canonical* choice. Then the procedure constructs a service automaton that encodes within its own structure only *canonical* choices instead of all possible choices. This procedure will be used to construct a canonical controller discussed in Chapter 4, Section 4.1.

In the following, we define a canonical choice of a given service R and the set $\alpha(q)$ of all *canonical choices* at state q of R.

Definition 3.27 (Canonical choices).
Let $\alpha(q)$ be the set of all choices at state q of service automaton R. Then, a *canonical choice* at state q is a choice $ch \in \alpha(q)$ that satisfies exactly one of the followings: either

– $ch = \{\}$; or

– for each $ch' \in \alpha(q)$ such that $ch \neq ch' \neq \{\}$ holds: either $(ch \subsetneq ch')$ or $(ch \cap ch' = \emptyset)$.

The set of all *canonical choices* at state q of R is denoted by $\widehat{\alpha}(q)$.

Intuitively, a canonical choice is a choice that cannot be reduced to any other choice in the same set by eliminating from it any single event that is contained in other choices.

In comparison to a Boolean formula $\alpha(q)$, the set of all choices $\alpha(q)$ at each state q of a service automaton is equivalent to the set of all valid assignments (each is a valid combination of events that satisfies an assignment) of the Boolean formula $\phi(q)$ at the same state q. Each valid assignment of the formula $\phi(q)$ is equivalent to one product expression of the formula in its disjunctive normal form. To this end, it is possible to find a minimal sum-of-products expression of the formula in a disjunctive normal form using a sum of *prime implicants* [see e. g., Micheli, 1994] of the formula. Therefore, a canonical choice of the set $\alpha(q)$ of all choices annotated at the same state q is equivalent to a prime implicant of $\alpha(q)$ in its disjunctive normal form.

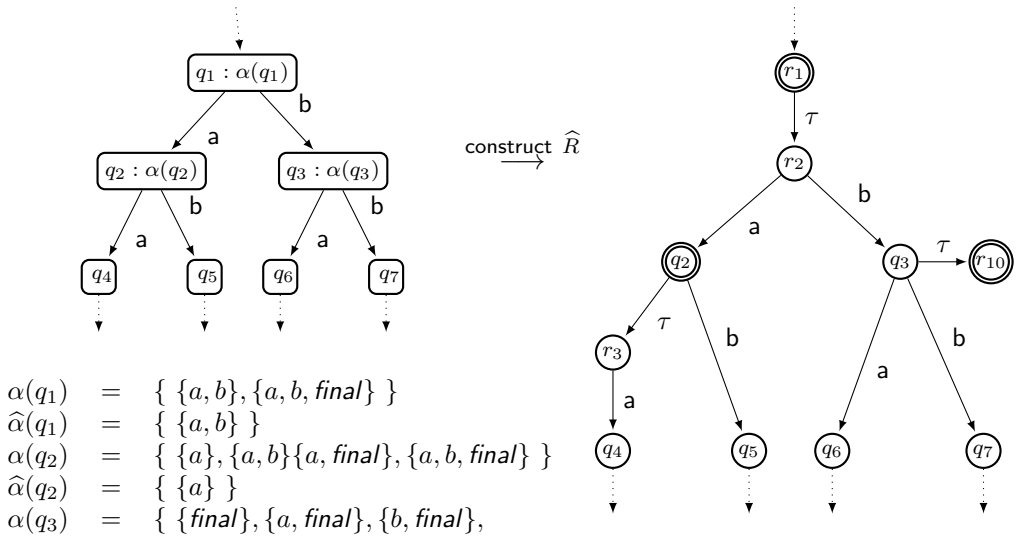
With the set of all canonical choices $\widehat{\alpha}(q)$ at state q, we denote the set of all events that are described by all canonical choices at state q with $\widehat{\Sigma}_\alpha(q)$.

Definition 3.28 (Canonical events).
Let $\widehat{\alpha}(q)$ be the set of all canonical choices at state q of a service automaton R. Then, the set $\widehat{\Sigma}_{\alpha(q)}$ of all canonical events at state q of R is defined as

$$\widehat{\Sigma}_{\alpha(q)} = \{m \in ch \mid ch \in \widehat{\alpha}(q)\}.$$

Example 3.29. The left hand side of Figure 3.7 shows three canonical choices $\widehat{\alpha}(q_1)$, $\widehat{\alpha}(q_2)$, and $\widehat{\alpha}(q_3)$, respectively derived from the choices $\alpha(q_1)$, $\alpha(q_2)$, and $\alpha(q_3)$ of states q_1, q_2, and q_3 of the annotate service automaton illustrated above.

The sets of all canonical events at state q_1, q_2, and q_3 are $\widehat{\Sigma}_\alpha(q_1) = \{a, b\}$, $\widehat{\Sigma}_\alpha(q_2) = \{a\}$, and $\widehat{\Sigma}_\alpha(q_3) = \{\mathit{final}\}$. ◁



$\alpha(q_1) = \{ \{a, b\}, \{a, b, \mathit{final}\} \}$
$\widehat{\alpha}(q_1) = \{ \{a, b\} \}$
$\alpha(q_2) = \{ \{a\}, \{a, b\}\{a, \mathit{final}\}, \{a, b, \mathit{final}\} \}$
$\widehat{\alpha}(q_2) = \{ \{a\} \}$
$\alpha(q_3) = \{ \{\mathit{final}\}, \{a, \mathit{final}\}, \{b, \mathit{final}\}, \{a, b\}, \{a, b, \mathit{final}\} \}$
$\widehat{\alpha}(q_3) = \{ \{\mathit{final}\} \}$

Figure 3.7.: Construction of $\widehat{R}$ from a choice service automaton.

Given a choice annotated service automaton R^α as a deterministic service automaton R where each state q is annotated by a choice $\alpha(q)$, we compute one specific element R^* that is described by $R^\alpha = [R, \varphi]$ from R^α by replacing each labeled state q with nondeterministic internal τ events between all canonical choices described at state q, and assigning all those events that are not canonical events to state q. The construction procedure of $\widehat{R}$ from a choice annotated service automaton R^α is defined as follows.

Definition 3.30 (Construction of $\widehat{R}$ from R^α).
Let $R = [Q, q_0, I, O, \rightarrow, \Omega]$ be a deterministic service automaton where each state $q \in Q$ is equipped with the set $\alpha(q)$ of all choices at q. Let $\widehat{\alpha}(q)$ be the set of all canonical choices at q and $\widehat{\Sigma}_{\alpha(q)}$ be the set of all canonical events at state q.

Then, the service $\widehat{R} = [Q^*, q_0^*, I, O, \rightarrow^*, \Omega^*]$ is defined as

- $Q^* = Q \ \cup \ \{q_{ch} \mid q \in Q \ \wedge \ ch \in \widehat{\alpha}(q)\}$;
- $q_0^* = q_0$;
- $\Omega^* = \{q_{ch} \mid \mathit{final} \in ch \in \widehat{\alpha}(q)\} \cup \{q \mid q \in Q \ \wedge \ \mathit{final} \notin \backslash\widehat{\Sigma}_{\alpha(q)}\}$;
- $\rightarrow^* = \{q \stackrel{\tau}{\rightarrow}^* q_{ch}\} \ \cup \{q_{ch} \stackrel{m}{\rightarrow}^* q' \mid q \stackrel{m}{\rightarrow} q' \ \wedge \ m \in ch \in \widehat{\alpha}(q)\}$.

Example 3.31. Figure 3.7 shows a fragment of service $\widehat{R}$ on the right hand side which is constructed from a fragment of a choice annotated service automaton R^α on the left hand side by applying Definition 3.30. ◁

In case of a strong choice annotated automaton R^α (where α is built on the set of actions including the internal τ event and R is a deterministic service automaton with τ loop at every state q of R^α), the following lemma shows that $\widehat{R}$ constructed from R^φ is described by R^φ with respect to the strong compliance relation.

Lemma 3.32 (Strong correspondence of $\widehat{R}$).
For each $\widehat{R}$ that is constructed from a strong choice annotated automaton R^α :

$$\widehat{R} \in \mathit{Comply}_\beta(R^\alpha).$$

Proof. Let $\widehat{R}$ be constructed from a strong choice annotated automaton $[R, \alpha]$ with $R = [Q, q_0, I, O, \rightarrow, \Omega]$ as described in Definition 3.30.

Let ϱ be a binary relation between states of $\widehat{R}$ and R with $\varrho \ = \ \{[q, q_R] \mid q \in Q \wedge \exists \beta \in \alpha(q) :: \beta = \mathit{enable}(q)\} \ \cup \ \{[q_\beta, q_R] \mid q_R \in Q_R \ \wedge \ \beta \in \alpha(q)\}$. As every state q in R has a τ-loop, and $\{\tau\} \in \varphi(q)$ holds, it follows that ϱ is a strong simulation relation and for each $[q, q_R] \in \varrho$ there exists $\beta \in \varrho(q)$ such that q strongly corresponds to β. Thus, $\widehat{R} \in \mathit{Comply}_\beta(R^\alpha)$ holds. □

In case of a structural choice annotated automaton R^α (where α is built on the set of actions excluding internal τ event and R is a deterministic service automaton without τ loop at every state q), the following lemma shows that $\widehat{R}$ constructed from R^α is described by R^α according to a structural compliance relation.

Lemma 3.33 (Structural correspondence of R^α).
For each $\widehat{R}$ that is constructed from a structural choice annotated automaton R^α :

$$\widehat{R} \in \mathit{Comply}_\gamma(R^\alpha).$$

Proof. Let $\widehat{R}$ be constructed from a structural choice annotated automaton $[R, \alpha]$ with $R = [Q, q_0, I, O, \rightarrow, \Omega]$ as described in Definition 3.30. Let ϱ be a binary relation between states of $\widehat{R}$ and R with $\varrho = \{[q^*, q] \mid q \in Q \wedge \exists \gamma \in \alpha(q) :: \gamma = \mathit{act}^*(\tau(q^*))\} \cup \{[q_\gamma, q] \mid q \in Q \wedge \gamma \in \alpha(q)\}$. As R is a deterministic and τ-free service automaton, it follows that ϱ is a structural simulation relation and for each $[q, q_R] \in \varrho$ there exists $\gamma \in \varrho(q)$ such that q structurally corresponds to β. Thus, $\widehat{R} \in \mathit{Comply}_\gamma(R^\alpha)$ holds. □

We will present the construction of a compact representative of the set of services to synthesize a compact canonical controller with experimental results in Chapter 4.

3.3. Finite Representation of Controllers

In this section, we present, for each compatibility criterion, an algorithm to construct a finite representation of all controllers of a given service. Each algorithm employs an annotated service automaton as a technique for representing the set of all controllers.

We first introduce in Section 3.3.1 the two concepts that are essential for characterizing sets of controllers; *situations* of one service and the *knowledge* that one service has about its communicating partners. Based on the two concepts, an algorithm for constructing a finite representation of all deadlock-free controllers is presented in Section 3.3.2 and an algorithm for constructing a finite representation of all responsive controllers is presented in Section 3.3.3. The last two sections presents the algorithms in a similar way, only using different notion for different compatibility criterion. The readers who are not interested in technical details may skip one of two sections.

Due to technical reasons, we consider the set $\mathbb{E}$ of *action events* according to different compatibility criteria under investigation, similarly to the previous sections.

– For deadlock freedom, we consider $\mathbb{E} = \Sigma \cup \{\tau\}$ as the set of action events.
– For responsiveness, we consider $\mathbb{E} = \Sigma$ as the set of action events.

For both cases, the set Σ is the set of communicating events that can be derived from the input and output message channels of a given service (see Section 2.1). We denote the set of all *action events* that includes a successfully terminating event τ as $\mathbb{E}_f = \mathbb{E} \cup \{\mathit{final}\}$.

3.3.1. Situations and Knowledge

In this section, we present two related concepts; a *situation* of one service and a *knowledge* that one service has about its communicating partner. These two concepts have been introduced in Lohmann et al. [2007a] for characterizing a deadlock of service composition from the viewpoint of one service participating in the service composition.

Consider the k-bounded composition of two interface compatible service automata S and P for message bound $k \in \mathbb{N}$ on each message channel. One state $[q_S, \mathcal{M}, q_P]$ of the composition $S \oplus P$ consists of a state q_S of S, a multiset $\mathcal{M}$ of all messages currently pending in the buffer between S and P, and a state q_P of P. A *situation* of S consists of its state q_S and the multiset $\mathcal{M}$ of currently pending messages. As each message that is currently pending in the buffer is bounded by k, $\mathcal{M}$ is finite. As S has also a finite number of states, then the set $situations(S)$ of all possible situations of S is also finite.

For a state $[q_S, \mathcal{M}, q_P]$ in $S \oplus P$, services S and P have mutual situations corresponding to $[q_S, \mathcal{M}, q_P]$. This means the situation of S corresponding to $[q_S, \mathcal{M}, q_P]$ is $[q_S, \mathcal{M}]$ whereas the situation of P corresponding to $[q_S, \mathcal{M}, q_P]$ is $[q_P, \mathcal{M}]$.

The knowledge of P about S is the mapping $\mathcal{K}_{(P,S)}$ from the set Q_P of all states of P to all situations of S defined as $\mathcal{K}_{(P,S)}(q_P) = \{\ [q_S, \mathcal{M}] \mid [q_S, \mathcal{M}, q_P] \in Q_{S \oplus P}\ \}$. This means that the knowledge of a state q_P of P collects all situations $[q_S, \mathcal{M}]$ of S such that $[q_S, \mathcal{M}, q_P]$ is a state of the composition $S \oplus P$.

Example 3.34. Figure 2.4 illustrates the composition $C \oplus A$ of Customer service C and Agency service A (from Figure 2.1). A situation of C that corresponds to state $L2 : [c2, [\text{req}], a1]$ is $[c2, [\text{req}]]$ whereas a situation of A that corresponds to state $L2 : [c2, [\text{req}], a1]$ is $[a1, [\text{req}]]$. The knowledge $\mathcal{K}_{(P,S)}(c2)$ at state $c2$ of Customer service C consists of all situations of A that corresponds to state $c2$ in the composition $C \oplus A$, that is, $\mathcal{K}_{(P,S)}(c2) = \{[a1, [\text{req}]], [a2, [\ ]], [a1, [\text{off}], a1], [a3, [\text{rej}]]\}$. ◁

Based on situations and knowledge, two *closure* and *event* operations on the set of knowledges are defined by Lohmann et al. [2007a] as follows.

Definition 3.35 (Closure, $closure(\mathcal{K})$).
Let $\mathcal{K} \subseteq situations(S)$ be a set of situations of service automaton S. Then, the *closure* of $\mathcal{K}$, denoted by $closure(\mathcal{K})$ is inductively defined as follows.
Basis: $\mathcal{K} \subseteq closure(\mathcal{K})$;
Step: if $[q, \mathcal{M}] \in closure(\mathcal{K})$ and $q \xrightarrow{e}_S q'$, then

- $[q', \mathcal{M} + m] \in closure(\mathcal{K})$ if $e = ?m$;
- $[q', \mathcal{M} - m] \in closure(\mathcal{K})$ if $e = !m$, and $m \in \mathcal{M}$; and
- $[q', \mathcal{M}] \in closure(\mathcal{K})$ if $e = \tau$.

The m-event operation that effects situations $\mathcal{K}$ is defined as follows.

Definition 3.36 (Event, $event(\mathcal{K}, m)$).
Let S and P be two service automata. Let $\mathcal{K} \subseteq situations(S)$ be a set of situations of service S and $e_P \in \Sigma_P \cup \{\tau\}$ be an event of P. Then, the e-event of $\mathcal{K}$ in S, denoted by $event(\mathcal{K}, e_P)$, is defined as

$$event(\mathcal{K}, e_P) = \begin{cases} \{[q, \mathcal{M} + m] \mid [q, \mathcal{M}] \in \mathcal{K}\}, & \text{if } e_P = !m, \\ \{[q, \mathcal{M} - m] \mid [q, \mathcal{M}] \in \mathcal{K}, m \in \mathcal{M}\}, & \text{if } e_P = ?m, \\ \mathcal{K}, & \text{if } e_P = \tau, \\ \emptyset, & \text{otherwise.} \end{cases}$$

For each behavioral compatibility criterion $\mathcal{B} \in \{df, rp\}$ that is investigated in this thesis, we assume the initial requirement that the composition must be bounded by k where $k \in \mathbb{N}$ is a messages bound on each asynchronous message channel. This means, no asynchronous message channel needs to store more than k pending messages during the communication between a service and each of its $\mathcal{B}$-controllers. As the service automaton model contains a finite number of states, clearly the set of all situations of a given service is also finite and bounded by $k \in \mathbb{N}$.

For a message bound $k \in \mathbb{N}$ and a given set $\mathcal{K}$ of situations, we first define a predicate to determine if a given event can *violate k-boundedness* of a given situation.

Definition 3.37 (k-violation, $violate_k(\mathcal{M}, m)$).
Let $\mathcal{M}$ be a multiset of currently pending message buffers between two service automata S and P. Then, for a message bound $k \in \mathbb{N}$, k-violation of event e, denoted by $violate_k(\mathcal{M}, e)$, is defined as

$$violate_k(\mathcal{M}, e) = \begin{cases} true, & \text{if } e = !m \text{ and } \mathcal{M}(m) > k, \\ false, & \text{otherwise.} \end{cases}$$

The predicate $violate_k(\mathcal{M}, e)$ is defined on message buffer $\mathcal{M}$ and message event m. It returns *true* in case $\mathcal{M}$ contains a situation in which e can violate the k-message bound, or returns *false* otherwise.

For each message bound $k \in \mathbb{N}$, we denote the set of all situations of service S as such that for each situation $\mathcal{M}$ in the set and each event $e \in \Sigma_S \cup \{final\}$ event e does not violate the bound k by $situations_k(S)$.

In order to calculate the *choices* from sets of situations of a given service, we introduce two additional predicates on a given situation; *step* and *activ*.

The first predicate *step* decides if the situation of a given service can make a move without any help from its partner (i. e., a service with compatible interface to a given service) and without violating the k-boundedness of message channels.

Definition 3.38 (Stepped events).
Let $[q, \mathcal{M}]$ be a situation of service automaton S and $e_S \in \Sigma \cup \{\tau, final\}$ be an event of service automaton S. Then, for a message bound $k \in \mathbb{N}$, the update of events e_S for situation $[q, \mathcal{M}]$, denoted by $step_k([q, \mathcal{M}], m_S)$, is defined as

$$step_k([q,\mathcal{M}],e_S) = \begin{cases} true, & \text{if either } (e_S = !m \text{ and } \mathcal{M}(m) < k) \\ & \quad \text{or } (e_S = ?m \text{ and } event(\{[q,\mathcal{M}]\},e_S) \neq \emptyset) \\ & \quad \text{or } (e_S = \tau \text{ and } event(\{[q,\mathcal{M}]\},e_S) \neq \emptyset) \\ & \quad \text{or } (e_S = final, q \in \Omega_S, \text{ and } \mathcal{M} = [\,]), \\ false, & \text{otherwise.} \end{cases}$$

For each situation $[q_S,\mathcal{M}]$ of S, the predicate $step_k([q_S,\mathcal{M}],m_S)$ either returns *true* in case event m_S of S can *update* its own situation $[q_S,\mathcal{M}]$ without violating the message bound $k \in \mathbb{N}$ and without any help from its partner (e. g., its interface compatible service P), or returns *false* otherwise.

Next, we define the predicate $activ_k$ for deciding whether or not an event m_P of P can be performed on the given situation $[q_S,\mathcal{M}]$ without violating the k-boundedness of message channels.

Definition 3.39 (Activated events).
Let S and P be two interface compatible service automata. Let $[q_S,\mathcal{M}]$ be a situation of S and $e_P \in \Sigma_P \cup \{\tau, final\}$ be an event of P. Then, for a message bound $k \in \mathbb{N}$, the activation of events e_P for situation $[q_S,\mathcal{M}]$, denoted by $activ_k([q_S,\mathcal{M}],e_P)$, is defined as

$$activ_k([q_S,\mathcal{M}],e_P) = \begin{cases} true, & \text{if either } (e_P = !m \text{ and } \mathcal{M}(m) < k) \\ & \quad \text{or } (e_P = ?m \text{ and } event(\{[q_S,\mathcal{M}]\},e_P) \neq \emptyset) \\ & \quad \text{or } (e_P = \tau \text{ and } event(\{[q_S,\mathcal{M}]\},e_P) \neq \emptyset) \\ & \quad \text{or } (e_P = final, q_S \in \Omega_S, \text{ and } \mathcal{M} = [\,]), \\ false, & \text{otherwise.} \end{cases}$$

For two interface compatible services S and P, the predicate $activ_k([q_S,\mathcal{M}],e_P)$ either returns *true* in case event e_P of P can *update* the situation $[q_S,\mathcal{M}]$ of S without violating the message bound $k \in \mathbb{N}$, or returns *false* otherwise.

3.3.2. Finite Representation of Deadlock-free Controllers

In this section, we present an algorithm for constructing a finite representation of deadlock-free controllers of a given service. We consider $\mathbb{E}_f = \Sigma \cup \{\tau, final\}$ as the set of all action events for the deadlock freedom compatibility criterion.

We first present a synthesis algorithm of one distinguished deadlock-free controller, called a *most permissive deadlock-free controller* [Lohmann et al., 2007a]. A *most permissive deadlock-free controller* of a given service is a deterministic deadlock-free controller that permits more behavior than other deadlock-free controllers. Based on a most permissive deadlock-free controller, we employ the technique described in Section 3.1.3 for representing the (possibly) infinite set of deadlock-free controllers of a service.

Most permissive Deadlock-free Controller

Lohmann et al. [2007a] and [Wolf, 2009a] have proposed algorithms for synthesizing a most permissive deadlock-free controller by overapproximating the behavior of all controllers. As a necessary condition, a most permissive deadlock-free controller *strongly simulates* every deadlock-free controller of a given service with respect to a given compatibility criterion. This means, this controller can be considered as a top element of the *strong simulation* preordered set of a service's controllers.

With the two operations *event* and *closure* on knowledge and situations previously introduced, we define the construction procedure of a *most permissive controller* $mp_{df,k}(S)$ (called the most permissive strategy in Lohmann et al. [2007a] and in Wolf [2009a]) of service S as follows.

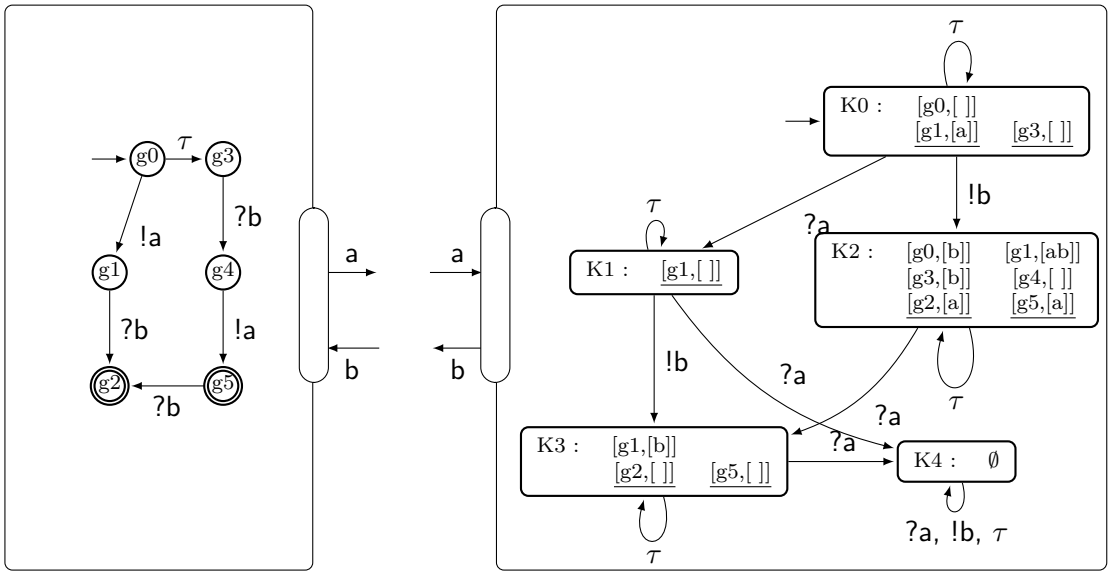
Definition 3.40 (Most permissive deadlock-free controller).
Let $k \in \mathbb{N}$ be a message bound on each message channel. The *most permissive k-deadlock-free* controller for a service automaton S is defined as the service automaton $mp_{df,k}(S) = [Q, q_0, I, O, \rightarrow, \delta]$ with

- $Q = \{q_{\mathcal{K}} \mid \mathcal{K} \subseteq situations_k(S)\}$,
- $q_0 = q_{\mathcal{K}_0}$ with $\mathcal{K}_0 = closure(\{[q_0, [\,]]\})$,
- $I = O_S, \quad O = I_S$,
- $\rightarrow = \{q_{\mathcal{K}} \xrightarrow{m} q_{\mathcal{K}'} \mid \mathcal{K}, \mathcal{K}' \subseteq situations_k(S), m \in \Sigma, \mathcal{K}' = closure(event(\mathcal{K}, m))\}$
  $\cup \{q_{\mathcal{K}} \xrightarrow{\tau} q_{\mathcal{K}} \mid \mathcal{K} \subseteq situations_k(S)\}$,
- $\Omega = \{q_{\mathcal{K}} \mid \mathcal{K} \subseteq situations_k(S), [q, [\,]] \in \mathcal{K}, q \in \Omega_S\}$.

A most permissive deadlock-free controller of a given service is deterministic by construction, as its transition relation is uniquely determined by the closure of all events starting from its initial state that represents the knowledge about situation of S at its initial state. A most permissive deadlock-free controller of a given service is also finite by construction, as each situation of S does not violate the message bound k. We illustrate the construction of a most permissive deadlock-free controller of a given service with the following example.

Example 3.41. Figure 3.8 shows service A_7 (taken from Figure 2.8(g)) on the left hand side and its most permissive k-deadlock-free controller $mp_{df,k}(A_7)$ for $k = 1$ on the right hand side constructed from A_7 by applying Definition 3.40.

Consider the most permissive k-deadlock-free controller $mp_{df,k}(A_7)$. There is no outgoing edge from state $K3$ with label $!b$, as $closure(event(K3, !b))$ produces a situation $[g1, [bb]]$ that violate message bound $k = 1$ of message channel b. As a result, the transition $g_5 \xrightarrow{?b}_{A_7} g_2$ in S_1 is never covered in the composition with its most permissive k-deadlock-free controller $mp_{df,k}(A_7)$ for message bound $k = 1$. ◁



(a) A_7 (b) $mp_{df,k}(A_7)$

Figure 3.8.: Service A_7 taken from Figure 2.8(g) and its most-permissive k-deadlock-free controller $mp_{df,k}(A_7)$ for $k = 1$.

In case a most permissive k-deadlock-free controller $mp_{df,k}(S)$ for service automaton S contains an empty set of states, the following proposition by Wolf [2009a] asserts that service automaton S is not k-deadlock-freely controllable for message bound $k \in \mathbb{N}$.

Proposition 3.42 [Wolf, 2009a].
For each message bound $k \in \mathbb{N}$ and each service S :

$$S \text{ is } k\text{-deadlock-freely controllable } \Leftrightarrow\ Q_{mp_{df,k}(S)} \neq \emptyset.$$

The following proposition asserts that the construction procedure defined by Definition 3.40 delivers a most permissive k-deadlock-free controller of service S for $k \in \mathbb{N}$ [Lohmann et al., 2007a].

Proposition 3.43 [Lohmann et al., 2007a].
For each message bound $k \in \mathbb{N}$ and each k-deadlock-freely controllable service S :

$$mp_{df,k}(S) \in df_k Controllers(S).$$

The following proposition from Lohmann et al. [2007a] asserts one necessary condition for a k-deadlock-free controller of service S; that is, each k-deadlock-freely controller P of S must be strongly simulated by a most permissive k-deadlock-free controller $mp_{df,k}(S)$ of S.

Proposition 3.44 [Lohmann et al., 2007a].
For each message bound $k \in \mathbb{N}$ and each k-deadlock-freely controllable service S:

$$P \in df_k Controllers(S) \;\Rightarrow\; mp_{df,k}(S) \text{ strongly simulates } P.$$

The simulation relation between most permissive deadlock-free controllers of two services is a necessary condition for deciding service substitutability. The following proposition [Stahl et al., 2009] asserts that if service T is a substitute for service S under k-deadlock freedom preorder, then a most permissive k-deadlock-free controller of T strongly simulates a most permissive k-deadlock-free controller of S.

Proposition 3.45 [Stahl et al., 2009].
For each message bound $k \in \mathbb{N}$ and each two interface equivalent services S and T:

$$T \sqsubseteq_{df,k} S \;\Rightarrow\; mp_{df,k}(T) \text{ strongly simulates } mp_{df,k}(S).$$

Deadlock-free Choices

In this section, we present a procedure to compute the *choices* for a most permissive deadlock-free controller of a given service. For each state of a given service, the set of all choices describes all possible combination of events that a communicating partner of service (i. e., a service with compatible interface to a given service) should offer in order to resolve all *stable situations* of a given service.

With the predicate *step* defined in Section 3.3.1, we define a *stable situation* as a situation that cannot make a move without any help from its communicating partner and without violating the k-boundedness of the message channels.

Definition 3.46 (Stable situations).
Let S and P be two interface compatible service automata and $[q_S, \mathcal{K}] \in \mathcal{K}$ be a situation of the set $\mathcal{K}$ of situations of S.

A *stable situation* $[q_S, \mathcal{M}]$ of $\mathcal{K}$ is a situation $[q_S, \mathcal{M}] \in \mathcal{K}$ such that for each $e_S \in enable(q_S)$ holds: $step_k([q_S, \mathcal{M}], e_S) = false$.

The set of all *stable situations* of $\mathcal{K}$ is denoted by $stable(\mathcal{K})$.

We employ the *stable* situations as defined above to calculate *deadlock-free choices* for a most permissive deadlock-free controller.

Informally, a deadlock-free choice of the set of situation $\mathcal{K}$ is a (non-empty) subset of all events in $\mathbb{E}_f = \Sigma \cup \{\tau, final\}$ that can resolve each stable situation in $\mathcal{K}$ and does not trigger any situation that eventually violates the message bound k.

For this purpose, we define, for a message bound $k \in \mathbb{N}$ and a given set $\mathcal{K}$ of situations, the set of all *legally updated events* of $\mathcal{K}$. That is, updating each situation in $\mathcal{K}$ by a

legally updated event in the set never yields a situation that violates the message bound k.

Definition 3.47 (Legally updated events, $\mathcal{L}_k(E,\mathcal{K})$).
Let $\mathcal{K}$ be a set of situations and E be a set of events and $\mathbb{E}_f$ be the set of events with $E \subseteq \mathbb{E}_f$. Then, the set of events in E that *legally update* $\mathcal{K}$ is defined as

$$\mathcal{L}_k(E,\mathcal{K}) = \{e \in E \mid \mathcal{K} = \emptyset \text{ or } \forall[q,\mathcal{M}] \in \mathit{closure}(\mathit{event}(\mathcal{K},e)) :: \mathit{violate}_k(\mathcal{M},e) = \mathit{false}\}.$$

With the set $\mathcal{L}_k(E,\mathcal{K})$ of events that legally update the situations $\mathcal{K}$, we define deadlock-free choices of a most permissive k-deadlock controller $mp_{df,k}(S)$ of service S as follows.

Definition 3.48 (Deadlock-free choices).
Let $k \in \mathbb{N}$ be a message bound and $\mathbb{E}_f$ be the set of events of a most permissive k-deadlock controller $mp_{df,k}(S)$ of service S. Let $q_\mathcal{K}$ be a state of $mp_{df,k}(S)$ with the set of situations $\mathcal{K}$ of S and the set $\mathcal{L}_k(\mathbb{E}_f,\mathcal{K})$ of all events in $\mathbb{E}_f$ that legally update $\mathcal{K}$. Then, the set of all *deadlock-free choices* of state $q_\mathcal{K}$ of S, denoted by $\varphi_k(q_\mathcal{K})$, is defined as :

$$\varphi_k(q_\mathcal{K}) = \begin{cases} \{ch \mid ch \subseteq \mathbb{E}_f\}, & \text{if } (\mathcal{K} = \emptyset \text{ or } \mathit{stable}(\mathcal{K}) = \emptyset), \\ \varphi'_k(q_\mathcal{K}), & \text{otherwise.} \end{cases}$$

where

$$\varphi'_k(q_\mathcal{K}) = \{ch \subseteq \mathcal{L}_k(\mathbb{E}_f,\mathcal{K}) \mid \forall[q_S,\mathcal{M}] \in \mathit{stable}(\mathcal{K}) : \exists e \in ch :: \mathit{activ}_k([q_S,\mathcal{M}],e) = \mathit{true}\}.$$

For each state $q_\mathcal{K}$, the set of choices at state $q_\mathcal{K}$ describes events that a compatible interface service of S should offer in order to resolve all *stable situations* of S in $\mathcal{K}$. The set $\mathcal{L}_k(\Sigma,\mathcal{K})$ of events that legally updates the situations $\mathcal{K}$ guarantees that a deadlock-free choice never describes an event that violates the k boundedness of message channels.

Example 3.49. Consider message bound $k = 1$ and a most permissive k-deadlock-free controller $mp_{df,k}(A_7)$ of A_7 from Figure 3.8, the activation of each event for all situations in $K0$ is illustrated in the following table. The underlined situations denote the stable situations of A_7 in which A_7 are in the states that cannot make a move without help from its partner (a service with compatible interface to A_7, e. g., $mp_{df,k}(A_7)$).

$[q,\mathcal{M}] \in$ K0	[g0, [ ]]	*[g1, [a]]*	*[g3, [ ]]*
$\mathit{activ}_k([q,\mathcal{M}],?a)$	*false*	*true*	*false*
$\mathit{activ}_k([q,\mathcal{M}],!b)$	*true*	*true*	*true*
$\mathit{activ}_k([q,\mathcal{M}],\tau)$	*true*	*true*	*true*
$\mathit{activ}_k([q,\mathcal{M}],\mathit{final})$	*false*	*false*	*false*

where
$stable(K0) = \{[g1, [a]], [g3, [\,]]\}$
$\mathbb{E}_f = \{?a, !b, \tau, final\}$
$\mathcal{L}_k(\mathbb{E}_f, K0) = \{?a, !b, \tau, final\}$
$\varphi(K0) = \{\ \{!b\}, \{\tau\}, \{?a, !b\}, \{!b, \tau\}, \{!b, final\}, \{?a, \tau\}, \{\tau, final\},$
$\{?a, !b, \tau\}, \{!b, \tau, final\}, \{?a, !b, final\}, \{?a, \tau, final\},$
$\{?a, !b, \tau, final\}\ \}.$

The k-deadlock-free choices of $K1$, $K2$, $K3$, and $K4$ are illustrated as follows.
$\varphi(K1) = \{\ \{!b\}, \{\tau\}, \{?a, !b\}, \{!b, \tau\}, \{!b, final\}, \{?a, \tau\}, \{\tau, final\},$
$\{?a, !b, \tau\}, \{!b, \tau, final\}, \{?a, !b, final\}, \{?a, \tau, final\},$
$\{?a, !b, \tau, final\}\ \}.$
$\varphi(K2) = \{\ \{?a\}, \{\tau\}, \{?a, \tau\}, \{?a, final\}, \{\tau, final\}, \{?a, \tau, final\}\ \}.$
$\varphi(K3) = \{\ \{final\}, \{\tau\}, \{final, \tau\}, \{?a, final\}, \{?a, \tau\}, \{\tau, final\}, \{?a, final, \tau\}\ \}.$
$\varphi(K4) = \{\ \{?a\}, \{!b\}, \{\tau\}, \{final\}, \{\ \},$
$\{?a, !b\}, \{?a, \tau\}, \{?a, final\}, \{!b, \tau\}, \{!b, final\}, \{\tau, final\},$
$\{?a, !b, \tau\}, \{!b, \tau, final\}, \{?a, !b, final\}, \{?a, \tau, final\},$
$\{?a, !b, \tau, final\}\ \}.$

Observe that $\{?a\}$ is not a choice for $K0$ as there is a stable situation $[g3, [\,]]$ in $K0$ where $?a$ is not an event that can activate the situation $[g3, [\,]]$. ◁

Representing all Deadlock-free Controllers

To represent the set of all deadlock-free controllers of a given service S, Lohmann et al. [2007a] and Massuthe [2009] have proposed a *deadlock-free operating guideline* of service S as the Boolean annotated automaton $OG(S) = [mp_{df,k}(S), \phi]$, where $mp_{df,k}(S)$ denotes the automaton of the most permissive deadlock-free controller of S. To analyze if service P is a k-deadlock-free controller of service S, one has to check if P matches with the Boolean annotated automaton $OG(S) = [mp_{df,k}(S), \phi]$. That is, P must have the same interface as $mp_{df,k}(S)$ and there must exist a strong simulation relation $\varrho \subseteq Q_P \times Q_{mp_{df,k}(S)}$ such that, for all $[q_P, q_m] \in \varrho$, the formula $\phi(q_m)$ is satisfied by q_P in the following assignment β. The assignment β is a Boolean function on $\Sigma \cup \{\tau, final\}$ such that $\beta(e)$, for $e \in \Sigma \cup \{\tau\}$, is true if there exists a state q'_P with $q_P \xrightarrow{e}_P q'_P$, and $\beta(final)$ is true if $q_P \in \Omega_P$.

In this thesis, we generalize operating guidelines to choice annotated service automata, i. e., choice annotated automaton $OG(S) = [mp_{df,k}(S), \varphi]$, where $mp_{df,k}(S)$ denotes the automaton of the most permissive deadlock-free controller of S and φ denotes the set of all deadlock-free choices at each state q_m of $mp_{df,k}(S)$. The set $\varphi(q_m)$ of all deadlock-free choice at state q_m of $mp_{df,k}(S)$ is equivalent to the set of all valid assignments of Boolean formulae $\phi(q_S)$ at state q_m of $mp_{df,k}(S)$ (cf. Section 3.1.3).

Example 3.50. Consider message bound $k = 1$, a k-deadlock-free operating guideline of service A_7 from Figure 3.8 is a most permissive k-deadlock-free controller $mp_{df,k}(A_7)$ from Figure 3.8, each state q is annotated by the set $\varphi(q)$ of choices at the corresponding state q as illustrated in Example 3.49. ◁

In the following lemma, we show that the most permissive k-deadlock-free controller $mp_{df,k}(S)$ of service S with each of its states $mp_{df,k}(S)$ annotated by the deadlock-free choices $\varphi(q_{\mathcal{K}})$ indeed represents the set of all k-deadlock-free controllers of S for message bound $k \in \mathbb{N}$ for each message channel.

Lemma 3.51 (Characterizing deadlock-free controllers).
For each message bound $k \in \mathbb{N}$ and each k-deadlock-freely controllable service S:

$$df_k\mathit{Controllers}(S) \;=\; \mathit{Comply}_\beta(mp_{df,k}(S)^\varphi).$$

Proof. We prove this lemma in two directions.

$\subseteq$: Suppose $P \in df_k\mathit{Controllers}(S)$. That is, $df_k(S \oplus P)$ holds by definition and we have $mp_{df,k}(S)$ strongly simulates P with ϱ (Proposition 3.44).

Consider $[q_P, q_m] \in \varrho$ with $[q_S, \mathcal{M}]$ as a stable situation of S at state q_m. Because $[q_P, q_m] \in \varrho$, $[q_S, \mathcal{M}]$ is also a stable situation of S at state q_P.

As $df_k(S \oplus P)$ holds, $[q_S, \mathcal{M}, q_P]$ is not a deadlock state in $S \oplus P$. This means state q_P must enable events that can resolve the stable situation $[q_S, \mathcal{M}]$. Because $[q_P, q_m] \in \varrho$ and $\varphi(q_m)$ describes all possible choices that can resolve every stable situation in $\mathcal{K}(q_m)$ and $[q_P, q_m] \in \varrho$, $[q_S, \mathcal{M}]$ is also stable at $q_{\mathcal{M}}$. This means, there exists a choice $ch \in \varphi(q_m)$ such that $ch = \mathit{enable}(q_P) = \beta \subseteq \Sigma \cup \{\mathit{final}, \tau\}$. That is, q_P strongly corresponds to $\varphi(q_m)$.

Thus, $P \in \mathit{Comply}(mp_{df,k}(S)^\varphi)$ holds.

$\supseteq$: Suppose $P \notin df_k\mathit{Controllers}(S)$. That is, $S \oplus P$ is not deadlock-free by definition.

Suppose $mp_{df,k}(S)$ strongly simulates P with ϱ and $[q_P, q_m] \in \varrho$. We will show that q_P does not strongly correspond to $\varphi(q_m)$.

Consider a deadlock state $[q_S, \mathcal{M}, q_P]$ in $S \oplus P$ with $[q_S, \mathcal{M}]$ as a stable situation of S. This means, the combination of all events that P offers at state q_P cannot resolve the stable situation $[q_S, \mathcal{M}]$ of S.

Because $\varphi(q_m)$ describes all possible choices that can resolve every stable situation in $\mathcal{K}(q_m)$ and because $[q_P, q_m] \in \varrho$, there exists a situation $[q_S, \mathcal{M}]$ of S that is stable at $q_{\mathcal{M}}$. This means that for each choice $ch \in \varphi(q_m)$ and $ch \subseteq \Sigma_P \cup \{\mathit{final}, \tau\}$ holds: $ch \neq \mathit{enable}(q_P)$. That is, q_P does not correspond strongly to $\varphi(q_m)$.

Thus, $df_k\mathit{Controllers}(S) = \mathit{Comply}_\beta(mp_{df,k}(S)^\varphi)$ holds. □

Lemma 3.51 illustrates that we can determine whether a service P is a k-deadlock-free controller of service S for $k \in \mathbb{N}$ by analyzing whether P strongly complies with $OG(S)$, denoted by $R \in \mathit{Comply}_\beta(mp_{df,k}(S)^\varphi)$. Service P *matches* with $mp_{df,k}(S)^\varphi$ if P has the same interface as $mp_{df,k}(S)$ and there is a strong simulation relation $\varrho \subseteq Q_P \times Q_{mp_{df,k}(S)}$ such that for all $[q_P, q_m] \in \varrho$: q_P strongly corresponds to q_m.

The operating guideline can also be used to determine whether a service T is a substitute for service S under both deadlock freedom preorder (i. e., $T \sqsubseteq_{df,k} S$) and equivalence (i. e., $T =_{df,k} S$). This decision procedure has been proposed by Stahl et al. [2009] and Stahl [2009] by relating the two Boolean annotaed service automata that represent the set of all deadlock-free controllers of S and T correspondingly.

In the following lemma, we generalize the decision procedure from Stahl et al. [2009] and Stahl [2009] and show that we can decide deadlock-free inclusion and equivalence of two services by comparing their most permissive deadlock-free controllers and their deadlock-free choices.

Lemma 3.52 (Deciding deadlock freedom inclusion).
For each message bound $k \in \mathbb{N}$ and each two interface equivalent services S and T:

$$T \sqsubseteq_{df,k} S \quad \Leftrightarrow \quad (mp_{df,k}(T) \text{ strongly simulates } mp_{df,k}(S) \text{ with } \varrho) \\ \wedge \ (\forall (q_{mT}, q_{mS}) \in \varrho : \varphi(q_{mS}) \subseteq \varphi(q_{mT})).$$

Proof. We prove this lemma in two directions.

$\Rightarrow$: Suppose $df_k Controllers(S) \subseteq df_k Controllers(T)$. This means that $mp_{df,k}(T)$ strongly simulates $mp_{df,k}(S)$ with ϱ (Proposition 3.45).

Consider $P \in df_k Controllers(S)$ and state q_P in P. This means, $mp_{df,k}(S)$ strongly simulates P with ϱ_S and there exists $(q_P, q_{mS}) \in \varrho_S$ with a choice $ch \in \varphi(q_{mS})$ such that q_P strongly corresponds to ch (Lemma 3.51).

Because $df_k Controllers(S) \subseteq df_k Controllers(T)$, $P \in df_k Controllers(T)$ follows. This means that $mp_{df,k}(T)$ strongly simulates P with ϱ_T and there exists also $[q_P, q_{mT}] \in \varrho_T$ and a choice $ch' \in \varphi(q_{mT})$ such that q_P strongly corresponds to the choice ch' (Lemma 3.51). It follows that $ch = ch' = enable(q_P)$.

Thus, $\varphi(q_{mS}) \subseteq \varphi(q_{mT})$ holds.

$\Leftarrow$: Suppose $P \in df_k Controllers(S)$ and $P \notin df_k Controllers(T)$.

Because $P \in df_k Controllers(S)$, $mp_{df,k}(S)$ strongly simulates P with ϱ_S (Proposition 3.44). Consider $[q_P, q_{mS}] \in \varrho_S$. It follows that there exists a choice in $\varphi_k(q_{mS})$ in which q_P strongly corresponds to (Lemma 3.51).

Suppose $mp_{df,k}(T)$ strongly simulates P with ϱ_T. Because $P \notin df_k Controllers(T)$, we consider $[q_P, q_{mT}] \in \varrho_T$. It follows that there exists no choice in $\varphi(q_{mT})$ in which q_P strongly corresponds to (3.51).

This means that there exists a choice that is in $\varphi(q_{mS})$ but not in $\varphi(q_{mT})$. Therefore, $\varphi(q_{mS}) \not\subseteq \varphi(q_{mT})$ holds.

Thus, this lemma holds. □

Corollary 3.53 (Deciding deadlock freedom equivalence).
For each message bound $k \in \mathbb{N}$ and each two interface equivalent services S and T:

$$T =_{df,k} S \;\Leftrightarrow\; (mp_{df,k}(T) \sim_{\text{bsim}} mp_{df,k}(S)) \wedge \forall (q_{mT}, q_{mS}) \in \sim_{\text{bsim}} : \varphi(q_{mS}) = \varphi(q_{mT})$$

where $\sim_{bsim}$ is a bisimulation relation between $mp_{df,k}(S)$ and $mp_{df,k}(T)$.

Proof. Because both $mp_{df,k}(S)$ and $mp_{df,k}(T)$ are deterministic service automata, the proof of this corollary follows from Lemma 3.52. □

As introduced by Definition 3.27 in Section 3.2.3, the concept of a *canonical choice* represents only a minimal subset of choices the preserves the same semantics of the original set of choices. Next, we define a *canonical deadlock-free choice* as an instance of Definition 3.27 for $\mathcal{B} = df_k$.

Definition 3.54 (Canonical deadlock-free choice).
Let $\varphi(q)$ be the set of all deadlock-free choices at state q of service automaton R. Then, a *canonical deadlock-free choice* at state q is a deadlock-free choice $ch \in \varphi(q)$ such that

– $ch = \{\}$; or

– for each $ch' \in \varphi(q)$ such that $ch \neq ch' \neq \{\}$ holds: either $(ch \subsetneq ch')$ or $(ch \cap ch' = \emptyset)$.

The set of all *canonical deadlock-free choices* at state q of R is denoted by $\widehat{\varphi}(q)$.

The first corollary shows that we can decide deadlock-free inclusion of two services by comparing their most permissive deadlock-free controllers and their corresponding sets of all canonical deadlock-free choices.

Corollary 3.55 (Deciding inclusion with canonical deadlock-free choices).
For each message bound $k \in \mathbb{N}$ and each two interface equivalent services S and T:

$$T \sqsubseteq_{df,k} S \;\Leftrightarrow\; (mp_{df,k}(T) \text{ strongly simulates } mp_{df,k}(S) \text{ with } \varrho) \wedge\; (\forall (q_{mT}, q_{mS}) \in \varrho : \widehat{\varphi}(q_{mT}) \subseteq \widehat{\varphi}(q_{mS})).$$

Proof. Follows from Lemma 3.52 and Definition 3.54. □

The next corollary shows that we can also decide deadlock-free inclusion equivalence of two services by comparing if their most permissive deadlock-free controllers are bisimilar and if the same set of all canonical deadlock-free choices at their bisimilar states are the same.

Corollary 3.56 (Deciding equivalence with canonical deadlock-free choices).
For each message bound $k \in \mathbb{N}$ and each two interface equivalent services S and T:

$$T =_{df,k} S \quad \Leftrightarrow \quad (mp_{df,k}(T) \sim_{\text{bsim}} mp_{df,k}(S)) \\ \wedge \ (\forall (q_{mT}, q_{mS}) \in \sim_{\text{bsim}} : \widehat{\varphi}(q_{mT}) = \widehat{\varphi}(q_{mS})).$$

Proof. Follows from Corollary 3.55 and Definition 3.54. □

3.3.3. Finite Representation of Responsive Controllers

In this section, we present an algorithm for constructing a finite representation of responsive controllers of a give services. We consider $\mathbb{E}_f = \mathbb{E} \cup \{final\} = \Sigma \cup \{final\}$ as the set of all action events for the responsiveness compatibility criterion.

We first distinguish *responsive choices* and *responsive valid choices* of a given service. Then we present an algorithm to synthesize one distinguished responsive controller, called a *most permissive responsive controller* as introduced by Wolf [2009a], Lohmann [2010].

Responsive Choices

In this section, we present a procedure to compute the *responsive choices* for a most permissive deadlock-free controller of a given service. For each state of a given service, the set of all choices describes all possible combinations of events that a partner of service should offer in order to resolve all stable situations of the given service.

We first distinguish a *wait situation* as a situation that either is *stable* (i. e., cannot make a move without help from its partner, cf. 3.3.2) or can form a local livelock without updating the message buffer between two services.

Definition 3.57 (Wait situations).
Let S and P be two interface compatible service automata and $[q_S, \mathcal{K}] \in \mathcal{K}$ be a situation of the set $\mathcal{K}$ of situations of S.

A situation $[q_S, \mathcal{M}] \in \mathcal{K}$ is a *wait situation* if $[q_S, \mathcal{M}]$ satisfies one of the followings: either

1. $[q_S, \mathcal{M}]$ is a stable situation, or
2. there exists $[q'_S, \mathcal{M}] \in \mathcal{K}$ such that q_S and q'_S are in the same stable τ-strongly connected component of S.

The set of all *wait situations* of $\mathcal{K}$ is denoted by $wait(\mathcal{K})$.

Every stable situation $[q_S, \mathcal{M}]$ of $\mathcal{M}$ is also by definition a wait situation of $\mathcal{M}$, i. e., $stable(\mathcal{K}) \subseteq wait(\mathcal{K})$ for a set $\mathcal{K}$ of situations, as stated in the first condition. A non-stable situation $[q_S, \mathcal{M}]$ is a wait situation, only if, there exists another situation $[q'_S, \mathcal{M}] \in \mathcal{K}$ such that states q_S and q'_S are in the same stable τ-strongly connected component of S, but S never updates the situation $\mathcal{M}$ by making such a move from state q_S to q'_S and

vice versa. Intuitively, this means, service S forms a local livelock while waiting for its partner to make a move.

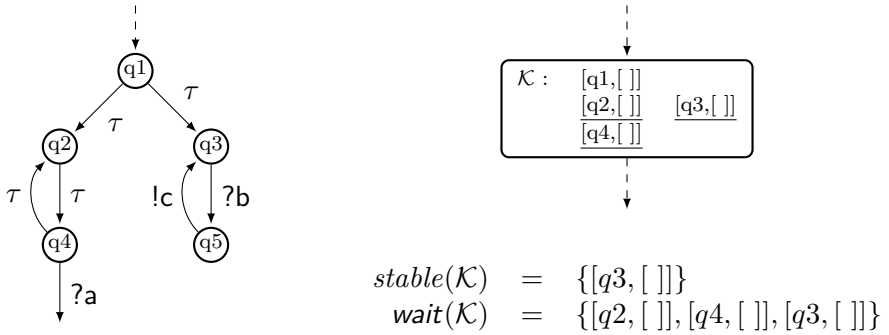


Figure 3.9.: Illustrations of stable situations and wait situations

Example 3.58. Figure 3.9 illustrates the difference between stable situations and wait situations. ◁

With the set of events that legally update situations $\mathcal{K}$, we define responsive choices of a most permissive k-responsive $mp_{rp,k}(S)$ of service S as follows.

Definition 3.59 (Responsive choices).
Let S and P be two interface compatible service automata. Let $k \in \mathbb{N}$ be a message bound on each message channel and $\mathbb{E}_f$ be the set of events of service P. Let $q_\mathcal{K}$ be a state of P with the set of respective situations $\mathcal{K}$ of S and $\mathcal{L}_k(\mathbb{E}_f, \mathcal{K})$ be the set of all events in $\mathbb{E}_f$ that legally update $\mathcal{K}$.

Then, the set of all *responsive choices* of state $q_\mathcal{K}$ of P, denoted by $\psi_k(q_\mathcal{K})$, is defined as

$$\psi_k(q_\mathcal{K}) = \begin{cases} \{ch \mid ch \subseteq \mathbb{E}_f\}, & \text{if } \mathcal{K} = \emptyset, \\ \psi'_k(q_\mathcal{K}), & \text{if } (\mathcal{K} \neq \emptyset \text{ and } wait(\mathcal{K}) \neq \emptyset), \\ \emptyset, & \text{otherwise.} \end{cases}$$

where

$$\psi'_k(q_\mathcal{K}) = \{ch \subseteq \mathcal{L}_k(\mathbb{E}_f, \mathcal{K}) \mid \forall [q_S, \mathcal{M}] \in wait(\mathcal{K}) : \exists e \in ch :: activ_k([q_S, \mathcal{M}], e) = true\}.$$

The set of all responsive choices can be computed locally at each state of a service automaton from the given set of situations of the service. Nevertheless, the responsiveness property of service composition cannot be guaranteed locally by checking the set of all responsive choices at a given state. Possibly there is an event of responsive choices of a given service that forces its interacting partner to perform a specific event and results in a deadlock situation. In case the service cannot perform any event to resolve the

deadlock situation, this yields an empty set of responsive choices. During the process of synthesizing a responsive controller, all paths of the service that lead to such as a situation must be removed.

For this purpose, we first define the *valid responsive choice* as a responsive choice in which every event of the choice is an enabled event at the current state.

Definition 3.60 (Valid responsive choices).
For each message bound $k \in \mathbb{N}$ on each message channel and each state $q_{\mathcal{K}}$ of a service automaton P, a responsive choice $ch \in \psi_k(q_{\mathcal{K}})$ is *valid* if for each event $e \in ch$ either $e = \mathit{final}$ or $e \in \mathit{enable}(q_{\mathcal{K}})$ holds. Otherwise, the responsive choice ch is *invalid*.

The set of all valid responsive choices is denoted by $\psi_k^*(\mathcal{K})$.

Obviously, $\psi_k^*(\mathcal{K}) \subseteq \psi_k(\mathcal{K})$ holds. Note that we consider a terminating event *event* as a valid event of a responsive choice.

Most Permissive Responsive Controller

Together with the two operations *event* and *closure* on knowledge and situations, we construct a *most permissive responsive controller* $mp_{rp,k}(S)$ of service S, representing the deterministic and finite-state responsive controller of S with a most permissive behavior.

Definition 3.61 (Most permissive responsive controller).
Let $k \in \mathbb{N}$ be a message bound on each message channel. The service automaton $T_k^0(S) = [Q, q_0, I, O, \rightarrow, \Omega]$ for a service automaton S is defined as

- $Q = \{q_{\mathcal{K}} \mid \mathcal{K} \subseteq \mathit{situations}_k(S)\}$,
- $q_0 = q_{\mathcal{K}_0}$ with $\mathcal{K}_0 = \mathit{closure}(\{[q_0, [\,]]\})$,
- $I = O_S, \quad O = I_S$,
- $\rightarrow = \{q_{\mathcal{K}} \xrightarrow{e} q_{\mathcal{K}'} \mid \mathcal{K}, \mathcal{K}' \subseteq \mathit{situations}_k(S), m \in \Sigma, \mathcal{K}' = \mathit{closure}(\mathit{event}(\mathcal{K}, e))\}$,
- $\Omega = \{q_{\mathcal{K}} \mid \mathcal{K} \subseteq \mathit{situations}_k(S), [q, [\,]] \in \mathcal{K}, q \in \Omega_S\}$.

Given $T_k^i(S)$ for $i \leq 0$, the service automaton $T_k^{i+1}(S)$ is obtained by removing state $q_{\mathcal{K}_i} \in Q_i$ if $\psi_k^*(\mathcal{K}_i) = \emptyset$. Thereby, the removal of a state includes the removal of its adjacent arcs and all states that becomes unreachable from the initial state q_0.

The most permissive responsive controller $mp_{rp,k}(S)$ for a service S is the service automaton $T_k^j(S)$ for the smallest j with $T_k^j(S) = T_k^{j+1}(S)$.

Example 3.62. Figure 3.10 illustrates the knowledge construction for Customer service C from Figure 2.1 for message bound $k = 1$. The underlined situations denote the wait

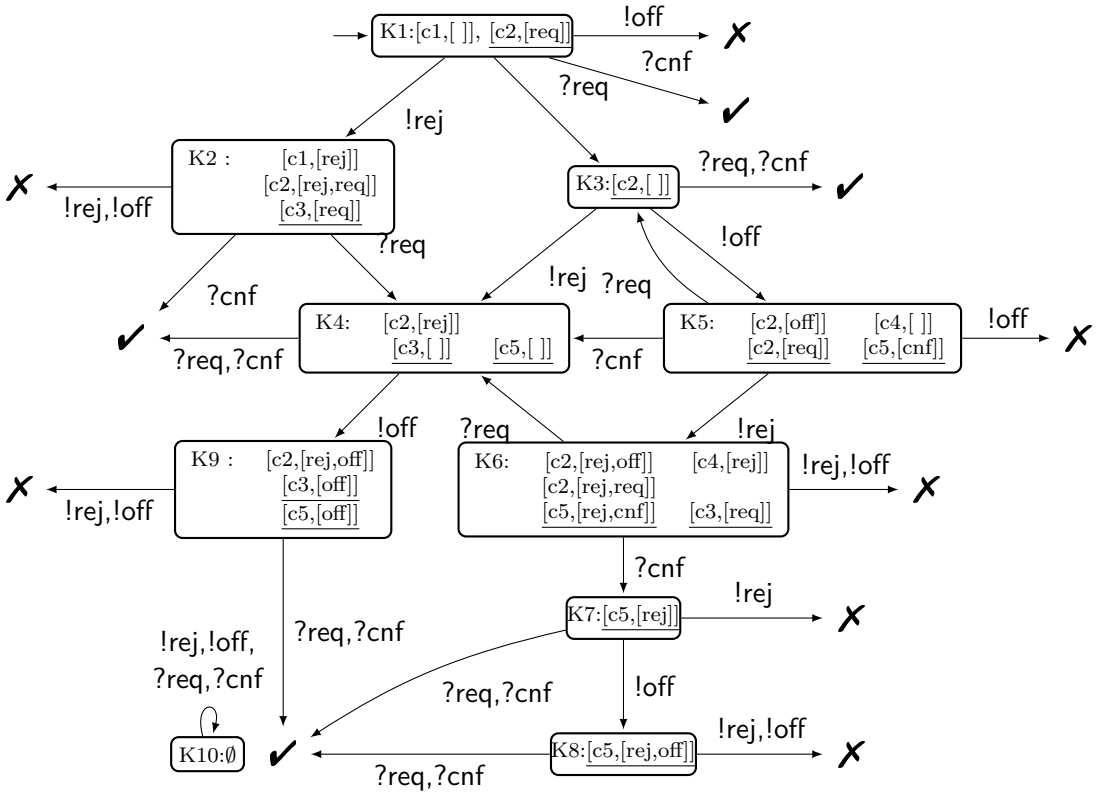


Figure 3.10.: Knowledge construction for Customer service C from Figure 2.1 for $k = 1$. Each state contains the set of corresponding situations of C in which an underlined situation denotes a wait situation of C.

situations of C. The transition with destination labeled by a cross mark means it is not a legal updated event of the knowledge at its source state.

Consider state $K8$ of the knowledge construction. The set of all wait situations at state $K8$ is $wait(K8) = \{$ [c5, [rej,off]] $\}$ and the activation of each event for all situations in $K8$ is illustrated in the following table.

$[q, \mathcal{M}] \in$ K8	*[c5, [rej,off]]*
$activ_k([q, \mathcal{M}], ?\text{req})$	*false*
$activ_k([q, \mathcal{M}], ?\text{cnf})$	*false*
$activ_k([q, \mathcal{M}], !\text{rej})$	*false*
$activ_k([q, \mathcal{M}], !\text{off})$	*false*
$activ_k([q, \mathcal{M}], final)$	*false*

This means, there is no responsive choice at $K8$, i. e., $\psi_k(K8) = \emptyset$, and there is also no valid responsive choice of $K8$, and therefore, $\psi_k^*(K8) = \emptyset$. As a result, state $K8$ and its adjacent transitions labeled with !off are removed during the construction of $mp_{rp,k}(C)$.

Consider $K7$. The set $\psi_k(K7)$ of all responsive choices at state $K7$ is $\psi_k(K7) = \{\{!off\}, \{!off, ?cnf\}, \{!off, ?req\}, \ldots, \{!off, ?cnf, ?req, \mathit{final}\}\}$. During the construction of $mp_{rp,k}(C)$, state $K7$ shall be removed as the result of removing state $K8$ and one of its adjacent transitions labeled with !off. This makes every choice of $\psi_k(K7)$ invalid as there is event !off in a choice where $!off \notin \mathit{enable}(K7)$. Therefore, $\psi_k^*(K7) = \emptyset$ in the next iteration of the process.

Consider $K6$. The set $\psi_k(K6)$ of all responsive choices at state $K6$ is $\psi_k(K6) = \{\{?req, ?cnf\}, \{?req, ?cnf, \mathit{final}\}\}$. During the construction of $mp_{rp,k}(C)$, state $K6$ shall be removed as the result of removing state $K7$ and one of its adjacent transitions labeled with ?cnf. This makes every choice of $\psi_k(K6)$ invalid as every choice of $\psi_k(K6)$ contains ?cnf. Therefore, $\psi_k^*(K6) = \emptyset$ in the next iteration of the process.

The set $\psi_k(K5)$ of all responsive choices at state $K5$ is $\psi_k(K5) = \{\{!rej\}, \{?req, ?cnf\}, \{?req, ?cnf, !off\}, \{?req, ?cnf, \mathit{final}\}, \ldots, \{!off, ?cnf, ?req, \mathit{final}\}\}$. As the transition labeled with !rej shall be removed during the construction of $mp_{rp,k}(C)$ as the result of removing state $K6$ and its adjacent transition. Nevertheless, not every responsive choice become invalid after removing $K6$ and its adjacent transitions. Some other responsive choices of $\psi_k(K5)$ are still valid, e. g., $\{?req, ?cnf\ \}$. This means $\psi_k(K5) \neq \psi_k^*(K5) \neq \emptyset$ in the next iteration of the process.

The set of all responsive choices at state $K9$ is also empty, therefore, state $K9$ and its adjacent arcs will also be removed during the procedure.

The most permissive k-responsive controller $mp_{rp,k}(C)$ of C for message bound $k = 1$ on each message channel is illustrated in Fig. 3.11. ◁

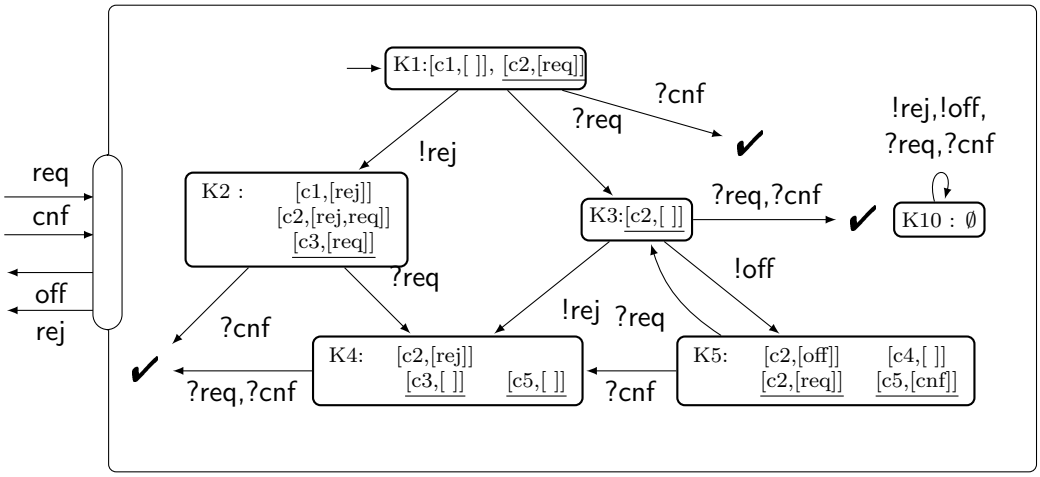


Figure 3.11.: Most-permissive responsive controller $rp\text{-}mp_k(C)$ of Customer service C from Figure 2.1 for $k = 1$.

Example 3.63. The valid k-responsive choices of $K1$, $K2$, $K3$, $K4$, $K5$, and $K10$ of the most permissive responsive controller $mp_{rp,k}(C)$ of C from Figure 3.11 are:

$\psi_k^*(K1) = \{\ \{?\text{req}\}, \{!\text{rej}\},$
$\{?\text{req}, !\text{rej}\}, \{?\text{req}, ?\text{cnf}\}, \{!\text{rej}, ?\text{cnf}\}, \{?\text{req}, \mathit{final}\}, \{!\text{rej}, \mathit{final}\},$
$\{?\text{req}, !\text{rej}, ?\text{cnf}\}, \{?\text{req}, !\text{rej}, \mathit{final}\}, \{?\text{req}, ?\text{cnf}, \mathit{final}\}, \{!\text{rej}, ?\text{cnf}, \mathit{final}\},$
$\{?\text{req}, !\text{rej}, ?\text{cnf}, \mathit{final}\}\ \}.$
$\psi_k^*(K2) = \{\ \{?\text{req}\}, \{?\text{req}, ?\text{cnf}\}, \{?\text{req}, \mathit{final}\}, \{?\text{req}, ?\text{cnf}, \mathit{final}\}\ \}.$
$\psi_k^*(K3) = \{\ \{!\text{rej}\}, \{!\text{off}\},$
$\{!\text{rej}, !\text{off}\}, \{!\text{rej}, ?\text{req}\}, \{!\text{rej}, ?\text{cnf}\}, \{!\text{rej}, \mathit{final}\}, \ldots,$
$\{!\text{rej}, !\text{off}, ?\text{req}, ?\text{cnf}\}, \ldots,$
$\{!\text{rej}, !\text{off}, ?\text{req}, ?\text{cnf}, \mathit{final}\}\ \}.$
$\psi_k^*(K4) = \{\ \{\mathit{final}\}, \{\mathit{final}, ?\text{req}\}, \{\mathit{final}, ?\text{cnf}\}, \{\mathit{final}, ?\text{req}, ?\text{cnf}\}\ \}.$
$\psi_k^*(K5) = \{\ \{?\text{req}, ?\text{cnf}\},$
$\{?\text{req}, ?\text{cnf}, !\text{off}\}, \{?\text{req}, ?\text{cnf}, \mathit{final}\}, \ldots,$
$\{?\text{req}, ?\text{cnf}, !\text{off}, \mathit{final}\}\ \}.$
$\psi_k^*(K10) = \{\ \{!\text{rej}\}, \{!\text{off}\}, \{?\text{req}\}, \{?\text{cnf}\}, \{\mathit{final}\}, \{\,\}$
$\{!\text{rej}, !\text{off}\}, \{!\text{rej}, ?\text{req}\}, \ldots,$
$\{!\text{rej}, !\text{off}, ?\text{req}\}, \{!\text{rej}, !\text{off}, ?\text{cnf}\}, \ldots,$
$\{!\text{rej}, !\text{off}, ?\text{req}, ?\text{cnf}\}, \{!\text{rej}, !\text{off}, ?\text{req}, \mathit{final}\}, \ldots,$
$\{!\text{rej}, !\text{off}, ?\text{req}, ?\text{cnf}, \mathit{final}\}\ \}.$
for message bound $k = 1$ on each message channel. ◁

In case a most permissive k-responsive controller $mp_{rp,k}(S)$ for service automaton S contains an empty set of states, the following proposition by Wolf [2009a] asserts that service automaton S is not k-responsively controllable for a message bound $k \in \mathbb{N}$ on each message channel.

Proposition 3.64 Wolf [2009a].
For each message bound $k \in \mathbb{N}$ and each service S :

$$S \text{ is } k\text{-responsively controllable } \Leftrightarrow\ Q_{mp_{rp,k}(S)} \neq \emptyset.$$

The following lemma asserts that the construction procedure defined by Definition 3.61 delivers a most permissive k-responsive controller of service S for a message bound $k \in \mathbb{N}$ on each message channel.

Lemma 3.65 (Most permissive responsive controller).
For each message bound $k \in \mathbb{N}$ and each k-responsively controllable service S :

$$mp_{rp,k}(S) \in rp_k Controllers(S).$$

Proof. We prove this lemma by showing that $rp_k(S \oplus mp_{rp,k}(S))$ holds.

Let $B_m = Beh_{S \oplus mp_{rp,k}(S)}(mp_{rp,k}(S))$ and $B_S = Beh_{S \oplus mp_{rp,k}(S)}(S)$. We will show that that B_m and B_S are responsive.

As $rp_k Controllers(S)$ is k-responsively controllable, $Q_{mp_{rp,k}(S)} \neq \emptyset$ (Proposition 3.64). Consider state q_m of $rp_k Controllers(S)$ and state $[q_S, \mathcal{M}, q_m]$ of the composition $S \oplus mp_{rp,k}(S)$.

Suppose $[q_S, \mathcal{M}]$ is a wait situation of S at state q_m. This means that there exists a stable τ-strongly connected component C in B_m that contains state $q = [q_S, \mathcal{M}, q_m]$. As $\psi_k^*(\mathcal{K}(q_m)) \neq \emptyset$ by construction of $rp_k Controllers(S)$, there exists a choice $ch \in \psi_k^*(\mathcal{K}(q_m))$ with $x \in enable(q_m)$, $x \neq \tau$, and $activ_k([q_S, \mathcal{M}], x) = true$ by definition. It follows that there is a state q_C in the component C in B_m with $x \in enable(q_C)$ and $x \neq \tau$. That is, $act^*(Q_C) \neq \emptyset$ holds and B_m is responsive follows.

Suppose C_S is a stable τ-strongly connected component in B_S with $act^*(C_S) = \emptyset$. We will show that $\psi_k^*(\mathcal{K}(q_m)) = \emptyset$, which contradicts to a condition on $mp_{rp,k}(S)$. As $act^*(C_S) = \emptyset$, C_S contains state q that is either a deadlock state or a divergent state (Lemma 2.12). Suppose $q = [q_S, \mathcal{M}, q_m]$. This means that any event x offered at state q_m in $mp_{rp,k}(S)$ (if any) does not resolve the situation $[q_S, \mathcal{M}]$ of $mp_{rp,k}(S)$. Therefore, for each choice $ch \in \psi_k^*(\mathcal{K}(q_m)$, there is no visible event at state $q_m \in \Sigma_m \cup \{final\}$ such that $m \in enable(q_m)$ and $m \in ch$ contributes to choice ch. This means, the set of valid responsive choice is empty (i. e., $\psi_k^*(\mathcal{K}(q_m) = \emptyset)$. This contradicts to the condition on $mp_{rp,k}(S)$ in which the set of all valid choices at every state of $mp_{rp,k}(S)$ must not be an empty set.

Thus, $rp_k(S \oplus mp_{rp,k}(S))$ holds and $mp_{rp,k}(S) \in rp_k Controllers(S)$ follows by definition. □

In the following corollary, we assert that every k-responsive controller P of S is structurally simulated by a most permissive k-responsive controller $mp_{rp,k}(S)$ of S.

Lemma 3.66 (Structural simulation of a responsive controller).
For each message bound $k \in \mathbb{N}$ and each k-responsively controllable service S:

$$P \in rp_k Controllers(S) \;\Rightarrow\; mp_{rp,k}(S) \text{ structurally simulates } P.$$

Proof. Suppose $rp_k(S \oplus P)$. This means both $B_P = Beh_{S \oplus P}(P)$ and $B_S = Beh_{S \oplus P}(S)$ are responsive by definition. Let $B_m = Beh_{S \oplus mp_{rp,k}(S)}(mp_{rp,k}(S))$. As $mp_{rp,k}(S) \in rp_k Controllers(S)$ (lemma 3.65), B_m is responsive by definition.

Consider a binary relation $\varrho \subseteq Q_P \times Q_m$ between states of P and states of $mp_{rp,k}(S)$. We will show that ϱ is a structural simulation relation.

Suppose $[q_{0P}, q_{0m}] \in \varrho$. By construction of $mp_{rp,k}(S)$, $\mathcal{K}'$ at q'_m is uniquely defined by $\mathcal{K}$ at q_m and $x \in \Sigma$, there is no other present x-labeled transition at q_m.

Consider $[q_P, q_m] \in \varrho$. For a state $[q_S, \mathcal{M}, q_P]$ in B_P, it follows that there exists also state $[q_S, \mathcal{M}, q_m]$ in B_m.

1. Consider the case where $q_P \xrightarrow{x}_P q'_P$ and $x \neq \tau$. As $mp_{rp,k}(S)$ and P have the same set of input and output message channels and the construction of $mp_{rp,k}(S)$ is driven by a closure of event of situations of S, there exists state q'_m in $mp_{rp,k}(S)$ with $\mathcal{K}'(q'_m) = closure(event(\mathcal{K}, x))$ and $q_m \xrightarrow{x} q'_m$ in $mp_{rp,k}(S)$.

If $closure(event(\mathcal{K}, x)) \neq \emptyset$, there exist transitions $[q_S, \mathcal{M}, q_P] \xrightarrow{x} [q_S, \mathcal{M}', q'_P]$ in B_P and $[q_S, \mathcal{M}, q_P] \xrightarrow{x} [q_S, \mathcal{M}', q'_m]$ in B_m. Otherwise, $[q_S, \mathcal{M}, q_P] \overset{x}{\not\rightarrow}$ holds in B_P and $[q_S, \mathcal{M}, q_m] \overset{x}{\not\rightarrow}$ holds in B_m.

It is not the case that $\psi^*_k(q_m) = \emptyset$, as $\psi^*_k(q_m) = \emptyset$ means for each $ch \in \psi_k(q_m)$ with $m \in ch$ that $x \notin enable(q_m)$ and $activ_k([q_S, \mathcal{M}], x) = false$; therefore, it is not possible to perform event x at state q_m. Then, state q'_m and transition $q_m \xrightarrow{x} q'_m$ shall be removed during the construction process of $mp_{rp,k}(S)$. Because B_P is also responsive, there is no outgoing transition from state $[q_S, \mathcal{M}, q_P]$ with label x.

Therefore, there exists state q'_m in $mp_{rp,k}(S)$ such that $(q'_P, q'_m) \in \varrho$.

2. Consider the case where $q_P \xrightarrow{\tau}_P q'_P$. As $mp_{rp,k}(S)$ is τ-free by construction and $rp_k(S \oplus P)$ holds by assumption, this means that there exists a transition $[q_S, \mathcal{M}, q_P] \xrightarrow{\tau} [q_S, \mathcal{M}, q'_P]$ in B_P and therefore $(q_P, q'_m) \in \varrho$ follows.

Thus, we conclude that $mp_{rp,k}(S)$ structurally simulates P. □

In the following corollary, we prove one necessary condition for a k-responsive preorder of service S; that is, its most permissive k-responsive controller must be strongly simulated by a most permissive k-responsive controller $mp_{rp,k}(S)$ of S.

Corollary 3.67 (Strong simulation of two most permissive responsive controllers).
For each message bound $k \in \mathbb{N}$ and each two interface equivalent services S and T:

$$T \sqsubseteq_{rp,k} S \quad \Rightarrow \quad mp_{rp,k}(T) \text{ strongly simulates } mp_{rp,k}(S).$$

Proof. Suppose $T \sqsubseteq_{rp,k} S$ holds, i. e., $rp_k Controllers(T) \supseteq rp_k Controllers(S)$.

Let $mp_{rp,k}(T)$ and $mp_{rp,k}(S)$ be two most permissive k-responsive controllers of S and of T respectively.

According to Lemma 3.65, we have $mp_{rp,k}(S) \in rp_k Controllers(S)$ holds and $mp_{rp,k}(S) \in rp_k Controllers(T)$ immediately follows from $rp_k Controllers(S) \subseteq rp_k Controllers(T)$.

According to Lemma 3.66, for every state q_{mS} of $mp_{rp,k}(S)$ there exists a state q_{mT} of $mp_{rp,k}(T)$ such that state q_{mS} is structurally simulated by state q_{mT} with a structural simulation relation $\varrho \subseteq Q_{mS} \times Q_{mT}$ between states of $mp_{rp,k}(S)$ and states of $mp_{rp,k}(T)$. Because both most permissive responsive controllers are deterministic and τ-free by construction, it follows that $mp_{rp,k}(T)$ strongly simulates $mp_{rp,k}(S)$. □

Representing all Responsive Controllers

To represent the set of all responsive controllers of a service, Lohmann [2010] has proposed *responsive operating guidelines* as an extension of *deadlock-free operating guidelines* [Lohmann et al., 2007a] and Massuthe [2009].

In this thesis, we generalize responsive operating guidelines to choice annotated service automata, i. e., choice annotated automaton $OG(S) = [mp_{rp,k}(S), \psi^*]$, where $mp_{rp,k}(S)$ denotes the automaton of the most permissive deadlock-free controller of S and ψ^* denotes the set of all valid responsive choices at each state q_m of $mp_{rp,k}(S)$. The set $\psi^*(q_m)$ of all valid responsive choices at state q_m of $mp_{rp,k}(S)$ is equivalent to the set of all valid assignments of Boolean formulae $\phi(q_S)$ at state q_m of $mp_{rp,k}(S)$.

Example 3.68. Consider message bound $k = 1$, a k-responsive operating guideline of Customer service C from Figure 2.1 is a most permissive k-responsive controller $mp_{rp,k}(C)$ from Figure 3.11, each state q is annotated by the set $\psi^*(q)$ of valid responsive choices at the corresponding state q as illustrated in Example 3.63. ◁

In the following lemma, we show that the most permissive k-responsive controller $mp_{rp,k}(S)$ of S with each of its state $mp_{rp,k}(S)$ annotated by the valid responsive choices $\psi^*(q_\mathcal{K})$ represents the set of all k-responsive controllers of S for message bound $k \in \mathbb{N}$.

Lemma 3.69 (Characterizing responsive controllers).
For each message bound $k \in \mathbb{N}$ and each k-responsively controllable services S:

$$rp_k Controllers(S) = Comply_\gamma(mp_{rp,k}(S)^{\psi^*}).$$

Proof. Let P be a service automaton with $I_S = O_P$ and $O_S = I_P$. Consider $B_P = Beh_{S\oplus P}(P)$ and $B_S = Beh_{S\oplus P}(S)$. We prove this lemma in two directions.

$\subseteq$: Suppose $P \in rp_k Controllers(S)$. That is, $rp_k(S \oplus P)$ holds by definition.

This means, a state $q = [q_S, \mathcal{M}, q_P]$ is not a deadlock state in $S \oplus P$ and the respective state q in B_S and of B_P is neither a deadlock state nor a divergent state in a stable τ-strongly connected component nor a non-stable state of B_S and of B_P (Lemma 3.65 and Lemma 2.12)

Because $rp_k(S \oplus P)$ holds, then we have $mp_{rp,k}(S)$ structurally simulates P with ϱ (Lemma 3.66). Consider $[q_P, q_m] \in \varrho$ and the three following cases:

1. $\mathcal{K}(q_m) \neq \emptyset$ and $wait(\mathcal{K}(q_m)) \neq \emptyset$:

   Consider a wait situation $[q_S, \mathcal{M}] \in wait(\mathcal{K}(q_m))$ of S at q_m and a stable τ-strongly connected component C_P in B_P which contains state $[q_S, \mathcal{M}, q_P]$. As $[q_S, \mathcal{M}, q_P]$ is neither a deadlock state nor a divergent state in C_P, there exists $x \in act^*(\tau(q_P))$ in P and $x \in enable([q_S, \mathcal{M}, q_P])$ in B_P.
   Consider $ch_P = \{x \mid x \in act^*(\tau(q_P))\}$. For each state q'_P that is internally reachable from q_P, it follows that there exists a structural simulation relation ϱ and state q_m in $mp_{rp,k}(S)$ such that $(q'_P, q_m) \in \varrho$ holds (Lemma 3.66). As $\psi^*_k(q_m) \neq \emptyset$ by construction, therefore, for a wait situation $[q_S, \mathcal{M}] \in wait(\mathcal{K}(q_m))$, there exists by construction at least one valid responsive choice in at state q_m. Because B_P is responsive, for each $x \in ch_P$ it follows that either $activ_k([q_S, \mathcal{M}], x) = true$ or $closure(event(\mathcal{K}, x)) = \emptyset$ must hold. This means that there exists a valid

responsive choice $ch \in \psi_k^*(q_m)$ at state q_m such that $ch = ch_P$ holds.
Consider state q_P'' with $q_P \stackrel{x}{\Rightarrow}_P q_P''$ such that q_P'' results from P performing event $x \in ch_P$ and q_P'' contributes to $[q_S, \mathcal{M}', q_P'']$. Either
(1) $[q_S, \mathcal{M}', q_P'']$ is contained in a stable τ-strongly connected component C_S in B_S. As B_S is responsive, $[q_S, \mathcal{M}', q_P'']$ is neither a deadlock state nor a divergent state in C_S, or
(2) there exists $q_S \stackrel{Y}{\Rightarrow}_S q_S'$ in S such that $[q_S', \mathcal{M}'', q_P'']$ is contained in a stable τ-strongly connected component C_S in B_S. As B_S is responsive, then $[q_S', \mathcal{M}'', q_P'']$ is neither a deadlock state nor a divergent state in C_S.

2. $\mathcal{K}(q_m) = \emptyset$: the state q_m is not reachable in $S \oplus mp_{rp,k}(S)$. Because $(q_P, q_m) \in \varrho$, this means $\mathcal{K}(q_P) = \emptyset$ holds and q_P is not reachable in $S \oplus P$. Therefore, either q_P can offer all possible events from $\Sigma_P \cup \{final\}$ ($ch \subseteq \Sigma_P \cup \{final\}$), or q_P is a divergent state ($ch = \{\}$), or q_P is a deadlock state ($ch = \{\}$).
3. $wait(\mathcal{K}(q_m)) = \emptyset$: this means that there is not wait situation in $\mathcal{K}(q_m)$ by definition. Such state does not contribute to a responsive choice and the state, if there is any, shall be removed by construction of $mp_{rp,k}(S)$. This case is not possible as it contradicts to the assumption $rp_k(S \oplus P)$.

For each case, we conclude that q_P structurally corresponds to $\psi_k^*(q_m)$. Thus, P matches with $mp_{rp,k}(S)^{\psi^*}$.

$\supseteq$: Suppose $P \notin rp_k Controllers(S)$. That is, either B_S or B_P is not responsive by definition. We will show that $P \notin Comply_\gamma(mp_{rp,k}(S)^{\psi^*})$ holds.

Suppose $mp_{rp,k}(S)$ structurally simulates P with ϱ and $(q_P, q_m) \in \varrho$. We will show that q_P does not correspond structurally to $\psi_k^*(q_m)$.

Consider the three following cases:

1. Assume a stable τ-strongly connected component C in B_S with $act^*(Q_C) = \emptyset$. Consider state $[q_S, \mathcal{M}, q_P]$ in Q_C. As B_S is not responsive, it follows that $[q_S, \mathcal{M}, q_P]$ is either a deadlock state or a divergent state in B_S (by definition of responsiveness and Lemma 2.12). As S is k-responsively controllable according to the assumption, this means that it is possible to resolve the situation $[q_S, \mathcal{M}]$. That is, for each event x that is offered by P and internally reachable from state q_P (i. e., $x \in act^*(\tau(q_P))$), it follows that x cannot resolve the wait situation $[q_S, \mathcal{M}]$. This means that $activ_k([q_S, \mathcal{M}], x) = false$ by definition. As $(q_P, q_m) \in \varrho$ holds, for each $ch \in \psi_k^*(q_m)$ and each $x \in ch \subseteq \Sigma_P \cup \{final\}$ holds: $activ_k([q_S, \mathcal{M}], x) = false$. Thus, q_P does not correspond structurally to $\psi_k^*(q_m)$.
2. Assume a stable τ-strongly connected component C in B_P with $act^*(Q_C) = \emptyset$. Consider state $[q_S, \mathcal{M}, q_P]$ in Q_C. As B_P is not responsive, it follows that $[q_S, \mathcal{M}, q_P]$ is either a deadlock state or a divergent state in B_P (by definition of responsiveness and Lemma 2.12). As S is k-responsively controllable according to the assumption, this means that it is possible to resolve the situation $[q_S, \mathcal{M}]$. That is, for each event x that is offered by P and is internally reachable from state q_P (i. e., $x \in act^*(\tau(q_P))$), it follows that x cannot resolve the wait

situation $[q_S, \mathcal{M}]$. This means that $activ_k([q_S, \mathcal{M}], x) = \mathit{false}$ by definition. As $(q_P, q_m) \in \varrho$ holds, for each $ch \in \psi_k^*(q_m)$ and each $x \in ch \subseteq \Sigma_P \cup \{\mathit{final}\}$ holds: $activ_k([q_S, \mathcal{M}], x) = \mathit{false}$. Thus, q_P does not correspond structurally to $\psi_k^*(q_m)$.

For each case, we conclude that q_P does not correspond structurally to $\psi_k^*(q_m)$. Therefore, $P \notin \mathit{Comply}_\gamma(mp_{rp,k}(S)^{\psi^*})$ follows.

Thus, $rp_k\mathit{Controllers}(S) = \mathit{Comply}_\gamma(mp_{rp,k}(S)^{\psi^*})$ holds. □

We can decide responsiveness inclusion and equivalence of two services by comparing their most permissive responsive controllers and their corresponding sets of all valid responsive choices of the most permissive responsive controllers.

Lemma 3.70 (Deciding responsiveness inclusion).
For each message bound $k \in \mathbb{N}$ and each two interface equivalent services S and T:

$$T \sqsubseteq_{rp,k} S \Leftrightarrow (mp_{rp,k}(T) \text{ structrally simulates } mp_{rp,k}(S) \text{ with } \varrho) \; . \\ \wedge \; (\forall (q_{mT}, q_{mS}) \in \varrho : \psi_k^*(q_{mS}) \subseteq \psi_k^*(q_{mT})).$$

Proof. We prove this lemma in two directions.

⇒ : Suppose $rp_k\mathit{Controllers}(S) \subseteq rp_k\mathit{Controllers}(T)$. This means that $mp_{rp,k}(T)$ structurally simulates $mp_{rp,k}(S)$ with ϱ (Corollary 3.67).

Consider $P \in rp_k\mathit{Controllers}(S)$ and state q_P in P. This means, $mp_{rp,k}(S)$ structurally simulates P with ϱ_S and there exists $(q_P, q_{mS}) \in \varrho_S$ with a choice $ch \in \psi_k^*(q_{mS})$ such that q_P structurally corresponds to ch (Lemma 3.69).

Because $rp_k\mathit{Controllers}(S) \subseteq rp_k\mathit{Controllers}(T)$ holds, $P \in rp_k\mathit{Controllers}(T)$ follows. This means that $mp_{rp,k}(T)$ structurally simulates P with ϱ_T and there exists also $(q_P, q_{mT}) \in \varrho_T$ and a choice $ch' \in \psi_k^*(q_{mT})$ such that q_P structurally corresponds to the choice ch' (Lemma 3.69). It follows that $ch = ch'$.

Thus, $\psi_k^*(q_{mS}) \subseteq \psi_k^*(q_{mT})$ holds.

⇐ : Suppose $P \in rp_k\mathit{Controllers}(S)$ and $P \notin rp_k\mathit{Controllers}(T)$.

Because $P \in rp_k\mathit{Controllers}(S)$, $mp_{rp,k}(S)$ strongly simulates P with ϱ_S (Lemma 3.66). Consider $(q_P, q_{mS}) \in \varrho_S$. It follows that there exists a choice in $\psi_k^*(q_{mS})$ in which q_P structurally corresponds to (Lemma 3.69).

Suppose $mp_{rp,k}(T)$ strongly simulates P with ϱ_T. Because $P \notin rp_k\mathit{Controllers}(T)$, we consider $(q_P, q_{mT}) \in \varrho_T$. It follows that there exists no choice in $\psi_k^*(q_{mT})$ in which q_P structurally corresponds to (Lemma 3.66).

This means that there exists a choice that is in $\psi_k^*(q_{mS})$ but not in $\psi_k^*(q_{mT})$. Therefore, $\psi_k^*(q_{mS}) \not\subseteq \psi_k^*(q_{mT})$ holds.

Thus, this lemma holds. □

Corollary 3.71 (Deciding responsiveness equivalence).
For each message bound $k \in \mathbb{N}$ and each two interface equivalent services S and T:

$$\begin{aligned} T =_{rp,k} S \;\Leftrightarrow\; & (mp_{rp,k}(T) \sim_{\text{bsim}} mp_{rp,k}(S)) \\ & \wedge\, \forall (q_{mT}, q_{mS}) \in \sim_{\text{bsim}} : \psi_k^*(q_{mS}) = \psi_k^*(q_{mT}) \end{aligned}$$

where $\sim_{bsim}$ is a bisimulation relation between $mp_{rp,k}(S)$ and $mp_{rp,k}(T)$.

Proof. Because both $mp_{rp,k}(S)$ and $mp_{rp,k}(T)$ are deterministic and τ-free service automata, the proof of this corollary follows from Lemma 3.70. □

We define a *canonical responsive choice* as an instance of Definition 3.27 for $\mathcal{B} = rp_k$.

Definition 3.72 (Canonical responsive choice).
Let $\psi^*(q)$ be the set of all responsive choices at state q of a service automaton R. Then, a *canonical responsive choice* at state q is a responsive choice $ch \in \psi^*(q)$ such that

– $ch = \{\}$; or

– for each $ch' \in \psi^*(q)$ such that $ch \neq ch' \neq \{\}$ holds: either $(ch \subsetneq ch')$ or $(ch \cap ch' = \emptyset)$.

The set of all *canonical responsive choices* at state q of R is denoted by $\widehat{\psi^*}(q)$.

The two following corollaries show that we can also decide responsive inclusion and equivalence of two services by comparing their most permissive responsive controllers and their corresponding sets of all canonical responsive choices.

Corollary 3.73 (Deciding responsiveness inclusion with canonical responsive choices).
For each message bound $k \in \mathbb{N}$ and each two interface equivalent services S and T:

$$\begin{aligned} T \sqsubseteq_{rp,k} S \;\Leftrightarrow\; & (mp_{rp,k}(T) \text{ strongly simulates } mp_{rp,k}(S) \text{ with } \varrho) \\ & \wedge\; (\forall (q_{mT}, q_{mS}) \in \varrho : \widehat{\psi^*}(q_{mT}) \subseteq \widehat{\psi^*}(q_{mS})). \end{aligned}$$

Proof. Follows from Lemma 3.70 and Definition 3.72. □

Corollary 3.74 (Deciding responsiveness equivalence with canonical responsive choices).
For each message bound $k \in \mathbb{N}$ and each two interface equivalent services S and T:

$$\begin{aligned} T =_{rp,k} S \;\Leftrightarrow\; & (mp_{rp,k}(T) \sim_{\text{bsim}} mp_{rp,k}(S)) \\ & \wedge\; (\forall (q_{mT}, q_{mS}) \in \sim_{\text{bsim}} : \widehat{\psi^*}(q_{mT}) = \widehat{\psi^*}(q_{mS})). \end{aligned}$$

Proof. Follows from Corollary 3.73 and Definition 3.72. □

3.4. Concluding Remarks

In this chapter, we introduced an *annotated service automaton* for representing a set of service automata. Previous work [e. g., Lohmann et al., 2007a, Massuthe, 2009, Stahl, 2009, Lohmann, 2010] has illustrated how to employ a Boolean annotated service automaton to construct a finite representation of all controllers of a given service, called an *operating guideline* of a given service for both deadlock freedom and responsiveness.

In this thesis, weabstracted from the Boolean formulae and annotated each state of an automaton with *choices*, representing a set of events representing possible combination of events. We demonstrated its equivalent characteristics to a Boolean annotated automata for representing a set of service automata. We introduced three operations that can be performed on choice annotated service automata. The first operation describes the intersection of two sets of services, where each set is represented by a choice annotated service automaton. The latter two operations construct different canonical representatives of the set characterized by an annotated service automaton. In the following chapter, we will investigate further a canonical representative that is constructed from an operating guideline of a given service introduced in this chapter.

Part II.

Theory

4. Canonical Representative of Controllers

In this chapter, we introduce one distinguished controller, called a *canonical controller*, of a given service. We study its distinguished properties and investigate a specific *refinement relation* between a canonical controller and any other controller of a given service. We propose a set of transformation rules each of which preserves either refinement or equivalence of the refinement relation as a method to synthesize from the canonical controller any other controller of the given service by means of transformation.

The chapter is organized as follows. Section 4.1 define a canonical controller of a given service for each behavioral compatibility criterion. Section 4.2 defines the classical *trace* semantics for service automata. Section 4.3 introduces a stable failure semantics for service automata and investigates the stable failures refinement of a canonical deadlock-free controller of a given service. Section 4.4 proposes responsive failures semantics for service automata and investigates the responsive failures refinement of a canonical responsive controller of a given service. Section 4.5 presents a set of transformation rules preserving refinement and/or equivalence of stable failures and/or responsive failures. Finally, Section 4.6 concludes the chapter.

4.1. Canonical Controller

In this section, we introduce one distinguished controller, called a *canonical controller*, of a given service. Intuitively, a canonical controller of a given service S is a controller of S that can canonically represent any other controller of service S.

The concept for synthesizing such a controller has been introduced by Mooij and Voorhoeve [2009] for the purpose of reasoning about an adapter for two services that are not deadlock-free controllers of one another. Thereby, Mooij and Voorhoeve [2009] have proposed to synthesize one distinguished controller from a finite representation of all controllers of S, also called an *operating guidelines* of S [see Section 3.3.2, Section 3.3.3, and Massuthe and Schmidt, 2005, Lohmann et al., 2007a, Massuthe, 2009], by encoding all possible choices as well as all sequences of events of all controllers within its own structure.

We present two variants of a canonical controller of a given service according to deadlock freedom criterion in Section 4.1.1 and two variants of a canonical controller of a given service according to responsiveness criterion in Section 4.1.2. We illustrate the comparison of the two variants of canonical controller with the experimental results for responsiveness criterion in Section 4.1.3.

4.1.1. Canonical Deadlock-free Controller

In this section, we present two variants of a *canonical deadlock-free controller* of a given service S as well as their construction procedures from a finite representation of all deadlock-free controllers of S, also known as a deadlock-free operating guideline of S (cf. Section 3.3.2).

A deadlock-free operating guideline of service S is an annotated service automaton that can compactly represent the set all deadlock-free controllers of S (cf. Lemma 3.51, Chapter 3). Without the annotated information, the underlying service of the operating guideline of a given service S is not a good candidate for representing the set of all controllers of S. Though the underlying service of the operating guideline of S, called a most permissive controller of S, is able to simulate every other controller of S, not every service simulated by the most permissive controller is a controller of S [Wolf, 2009a].

For each message bound $k \in \mathbb{N}$ and a k-deadlock-free operating guideline of service S, we construct one (complete) representative, denoted by $\mathcal{C}_{df,k}(S)$, of the set of all k-deadlock-free controllers of S that is represented by the given operating guidelines. For this purpose, we employ the procedure described in Section 3.2.2. The service $\mathcal{C}_{df,k}(S)$ is constructed from the operating guideline by replacing each of its state q with a fragment of nondeterministic internal τ events between all deadlock-free choices that are described at state q. We denote the constructed service $\mathcal{C}_{df,k}(S)$ as a *canonical k-deadlock-free controller* of service S.

Definition 4.1 (Canonical k-deadlock-free controller $\mathcal{C}_{df,k}$).
Let $k \in \mathbb{N}$ be a message bound for each message channel. Let $mp_{df,k}(S)$ be a most

permissive deadlock-free controller of a k-deadlock-freely controllable service S where each state $q \in Q$ is equipped with the set $\varphi(q)$ of deadlock-free choices at state q.

Then, a canonical k-deadlock-free controller $\mathcal{C}_{df,k}(S)$ of service S is the service constructed from $mp_{df,k}(S)^{\varphi}$ as defined by Definition 3.23.

In the following corollary, we show that $\mathcal{C}_{df,k}(S)$ is indeed a k-deadlock-free controller of service S.

Corollary 4.2 ($\mathcal{C}_{df,k}(S)$ is a k-deadlock-free controller of S).
For each message bound $k \in \mathbb{N}$ and each k-deadlock-freely controllable service S :

$$\mathcal{C}_{df,k}(S) \in df_k Controllers(S).$$

Proof. Let $[mp_{df,k}(S), \varphi]$ be an annotated most permissive k-deadlock-free controller of S and $\mathcal{C}_{df,k}(S)$ be constructed from $[mp_{df,k}(S), \varphi]$ as described in Definition 4.1. By construction of $\mathcal{C}_{df,k}(S)$, we obtain $\mathcal{C}_{df,k}(S) \in Comply_{\beta}(mp_{df,k}(S)^{\varphi})$ (Lemma 3.25). Because $df_k Controllers(S) = Comply_{\beta}(mp_{df,k}(S)^{\varphi})$ (Lemma 3.51), then $\mathcal{C}_{df,k}(S) \in df_k Controllers(S)$ follows. □

The construction of $\mathcal{C}_{df,k}(S)$ employs the procedure of constructing a complete representative of the set of services as previously mentioned in Section 3.2.2. As a result, $\mathcal{C}_{df,k}(S)$ can be fairly large in size. To synthesize a more compact representative of the set of all deadlock-free controllers, we can also employ an alternative construction procedure as defined in Section 3.2.3. We denote this compact representative $\widehat{\mathcal{C}}_{df,k}(S)$ as a *compact canonical k-deadlock-free controller* of service S.

Definition 4.3 (Compact canonical k-deadlock-free controller $\widehat{\mathcal{C}}_{df,k}$).
Let $k \in \mathbb{N}$ be a message bound for each message channel. Let $mp_{df,k}(S)$ be a most permissive deadlock-free controller of a k-deadlock-freely controllable service S where each state $q \in Q$ is equipped with the set $\varphi(q)$ of deadlock-free choices at state q.

Then, a compact canonical k-deadlock-free controller $\widehat{\mathcal{C}}_{df,k}(S)$ of service S is the service constructed from $mp_{df,k}(S)^{\varphi}$ as defined by Definition 3.30.

The following corollary shows that $\widehat{\mathcal{C}}_{df,k}(S)$ is a k-deadlock-free controller of service S.

Corollary 4.4 ($\widehat{\mathcal{C}}_{df,k}(S)$ is a k-deadlock-free controller of S).
For each message bound $k \in \mathbb{N}$ and each k-deadlock-freely controllable service S :

$$\widehat{\mathcal{C}}_{df,k}(S) \in df_k Controllers(S).$$

Proof. Let $[mp_{df,k}(S), \varphi]$ be an annotated most permissive k-deadlock-free controller of S, and $\mathcal{C}_{df,k}(S)$ be constructed from $[mp_{df,k}(S), \varphi]$ as described in Definition 4.1. By construction of $\mathcal{C}_{df,k}(S)$, we obtain $\widehat{\mathcal{C}}_{df,k}(S) \in \mathit{Comply}_\beta(mp_{df,k}(S)^\varphi)$ (Lemma 3.32). Because $\mathit{df}_k\mathit{Controllers}(S) = \mathit{Comply}_\beta(mp_{df,k}(S)^\varphi)$ (Lemma 3.51), $\widehat{\mathcal{C}}_{df,k}(S) \in \mathit{df}_k\mathit{Controllers}(S)$ follows. □

The following examples show four service automata and the construction of a canonical deadlock-free controller service from a deadlock-free operating guideline of one service automaton of the four.

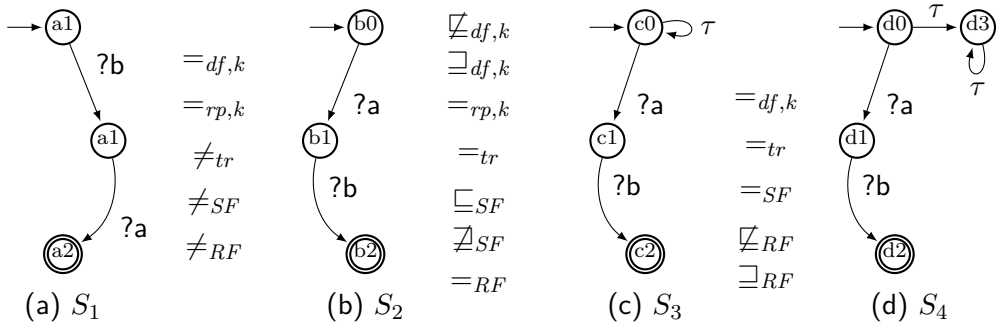


Figure 4.1.: Services S_1, S_2, S_3, and S_4, each of which has interface $I = \{a, b\}$ and $O = \{\}$. For message bound $k = 1$ on each message channel.

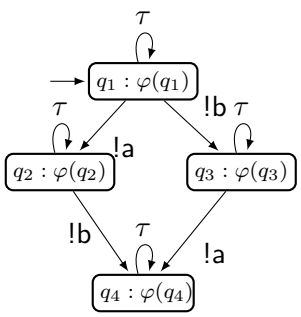
Example 4.5. Figure 4.1 illustrates four service automata; S_1, S_2, S_3, and S_4 with equivalent interface ($I = \{a, b\}$ and $O = \{\}$). For readability purpose, the interface in each service is omitted from the figure.

Consider $k = 1$. Every service in the figure is k-deadlock-freely controllable. Services S_1 and S_2 are equivalent under k-deadlock freedom ($S_1 =_{df,k} S_2$). Similarly, service S_3 and S_4 are equivalent under k-deadlock freedom ($S_3 =_{df,k} S_4$). Nevertheless, service S_2 and S_3 are not equivalent under k-deadlock freedom. But we can substitute service S_1 (or S_2) by service S_3 under k-deadlock freedom ($S_2 \sqsupseteq_{df,k} S_3$).

Figure 4.2 illustrates on the top a k-deadlock-free operating guideline $mp_{df,k}(S_1)^\varphi$ of S_1 and on the left hand side a k-deadlock-free controller $\mathcal{C}_{df,k}(S_1)$ of S_1 constructed from the operating guideline. The constructed $\mathcal{C}_{df,k}(S_1)$ has the interface $I = \{\}$ and $O = \{a, b\}$ (omitted from the figure). That is, the interface $\mathcal{C}_{df,k}(S_1)$ is compatible with the interface of S_1, S_2, S_3, and S_4.

Figure 4.2 illustrates on the right hand side the constructed $\widehat{\mathcal{C}}_{df,k}(S_1)$ from an operating guideline of S_1 from Figure 4.1. The constructed $\mathcal{C}_{df,k}(S_1)$ has the interface $I = \{\}$ and $O = \{a, b\}$ (omitted from the figure).

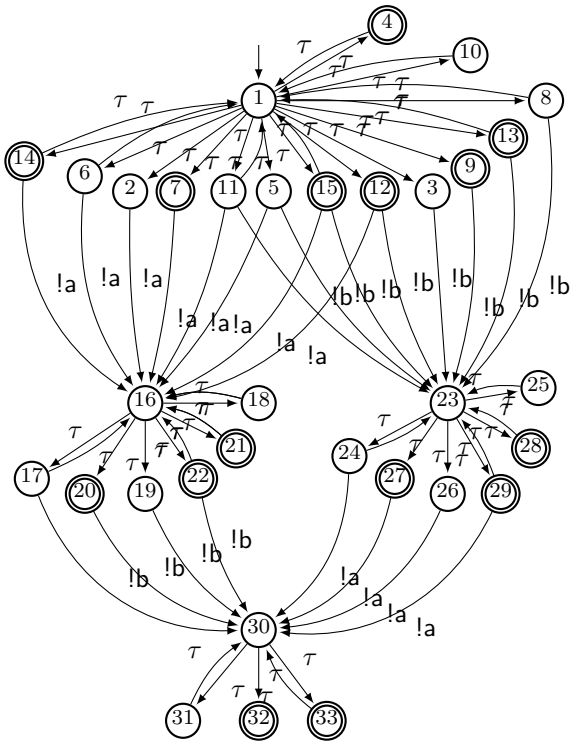
In comparison to $\mathcal{C}_{df,k}(S_1)$ on the left hand side, the synthesized controller $\widehat{\mathcal{C}}_{df,k}(S_1)$ is relatively smaller in size, as the construction procedure of $\widehat{\mathcal{C}}_{df,k}(S_1)$ replaces each state q of the operating guideline with a fragment of non-deterministic τ events of only canonical choices at state q, but keep other events that are not described by a canonical choice at



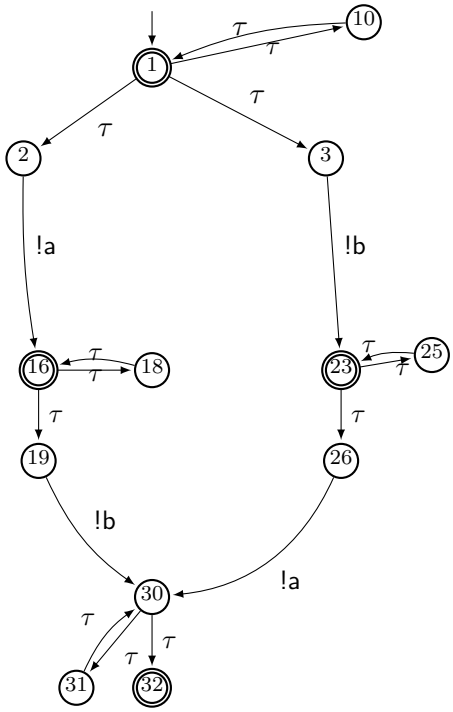
$$
\begin{aligned}
\varphi(q_1) &= \{\ \{!a\},\{!b\},\{\tau\},\\
&\quad \{!a,!b\},\{!a,\mathit{final}\},\{!a,\mathit{final}\},\{!b,\tau\},\\
&\quad \{!b,\mathit{final}\},\{\tau,\mathit{final}\},\{!a,!b,\tau\},\{!a,\tau,\mathit{final}\},\\
&\quad \{!a,!b,\mathit{final}\},\{!b,\tau,\mathit{final}\},\{!a,!b,\tau,\mathit{final}\}\ \}\\
\varphi(q_2) &= \{\ \{!b\},\{\tau\},\\
&\quad \{!b,\tau\},\{!b,\mathit{final}\},\{\tau,\mathit{final}\},\{!b,\tau,\mathit{final}\}\ \}\\
\varphi(q_2) &= \{\ \{!a\},\{\tau\},\\
&\quad \{!a,\tau\},\{!a,\mathit{final}\},\{\tau,\mathit{final}\},\{!a,\tau,\mathit{final}\}\ \}\\
\varphi(q_4) &= \{\ \{\mathit{final}\},\{\tau\},\\
&\quad \{\mathit{final},\tau\}\ \}
\end{aligned}
$$

(a) $mp_{df,k}(S_1)^{\varphi}$

construct $\mathcal{C}_{df,k}(S_1)$ ↙ ↘ construct $\widehat{\mathcal{C}}_{df,k}(S_1)$



(b) $\mathcal{C}_{df,k}(S_1)$



(c) $\widehat{\mathcal{C}}_{df,k}(S_1)$

Figure 4.2.: A k-deadlock-free operating guidelines $mp_{df,k}(S_1)^{\varphi}$ of S_1 (from Figure 4.1) and two canonical k-deadlock-free controllers; $\mathcal{C}_{df,k}(S_1)$ and $\widehat{\mathcal{C}}_{df,k}(S_1)$ of S_1, both with $I = \{\ \}$ and $O = \{a, b\}$ for message bound $k = 1$.

the respective state of q in $\widehat{\mathcal{C}}_{df,k}(S_1)$. In this example, at each state q_1, q_2, and q_3 of the operating guideline, there is only one *final* event that is not described by the canonical choice at the respective state of q_1, q_2, and q_3, and this *final* event which is modeled by a final state of service automaton at state 1, 16, and 23 respectively. For state q_4 of the operating guideline, every possible event at state q_4 (these are *final* and τ) is described by a canonical choice at q_4. ◁

We wil discuss some properties of the two canonical deadlock-free controllers in Section 4.2 and Section 4.3.

4.1.2. Canonical Responsive Controller

In this section, we present two variants of the *canonical responsive controller* of a given service S and their construction procedure from a finite representation of all responsive controllers of S, also known as a responsive operating guideline of S (cf. Section 3.3.3).

A responsive operating guideline of service S is an annotated service automaton that can compactly represent the set of all responsive controllers of S (cf. Lemma 3.51). Similarly to a deadlock-free operating guideline, a responsive operating guideline is not a pure service but a service annotated with extra information. This means that, any operation on service is not directly applicable to the responsive operating guideline. Though the underlying service of the responsive operating guideline of S is a most permissive responsive controller of S and it is able to simulate every other responsive controller of S, not every service simulated by the most permissive responsive controller is a responsive controller of S [Wolf, 2009a].

From each message bound $k \in \mathbb{N}$ and a k-responsive operating guideline of service S, we construct one (complete) representative, denoted by $\mathcal{C}_{rp,k}(S)$, of the set of all k-responsive controllers of service S that is represented by the given operating guideline. Similar to the construction procedure of the $\mathcal{C}_{df,k}(S)$ introduced in the previous section, we employ the procedure described in Section 3.2.2. The service $\mathcal{C}_{rp,k}(S)$ is constructed from the operating guideline by replacing each state q with a fragment of of nondeterministic internal τ events between all valid responsive choices that are described at state q. We denote the constructed service $\mathcal{C}_{rp,k}(S)$ as a *canonical k-responsive controller* of service S.

Definition 4.6 (Canonical k-responsive controller $\mathcal{C}_{rp,k}$).
Let $k \in \mathbb{N}$ be a message bound for each message channel and let $mp_{rp,k}(S)$ be a most permissive responsive controller of a k-responsively controllable service S where each state $q \in Q$ is equipped with the set $\psi^*(q)$ of valid responsive choices at state q.

Then, a canonical k-responsive controller $\mathcal{C}_{rp,k}(S)$ of service S is the service constructed from $mp_{rp,k}(S)^{\psi^*}$ as defined by Definition 3.23.

In the following corollary, we show that $\mathcal{C}_{rp,k}(S)$ is indeed a k-deadlock-free controller of service S.

Corollary 4.7 ($\mathcal{C}_{rp,k}(S)$ is a k-responsive Controller of S).
For each message bound $k \in \mathbb{N}$ and each k-responsively controllable service S :

$$\mathcal{C}_{rp,k}(S) \in rp_k Controllers(S).$$

Proof. Let $(mp_{rp,k}(S), \psi^*)$ be an annotated most permissive k-responsive controller of S with $mp_{rp,k}(S)$ and $\mathcal{C}_{rp,k}(S)$ be constructed from $(mp_{rp,k}(S), \psi^*)$ as described in Definition 4.6. By construction of $\mathcal{C}_{rp,k}(S)$, we obtain $\mathcal{C}_{rp,k}(S) \in Comply_\gamma(df\text{-}mp_k(S)^{\psi^*})$ (Lemma 3.26). Because $rp_k Controllers(S) = Comply_\gamma(rp\text{-}mp_k(S)^{\psi^*})$ (Lemma 3.69), then $\mathcal{C}_{rp,k}(S) \in rp_k Controllers(S)$ follows. □

The construction of $\mathcal{C}_{rp,k}(S)$ employs the procedure of constructing a complete representative of the set of services as previously mentioned in Section 3.2.2. As a result, $\mathcal{C}_{rp,k}(S)$ can be large in size. To synthesize a more compact representative of the set of all responsive controllers, we can also employ an alternative construction procedure as defined in Section 3.2.3. We denote We denote this compact representative $\mathcal{C}_{rp,k}(S)$ as a *compact canonical k-responsive controller* of service S.

Definition 4.8 (Compact canonical k-responsive controller $\widehat{\mathcal{C}}_{rp,k}(S)$).
Let $k \in \mathbb{N}$ be a message bound for each message channel. Let $mp_{rp,k}(S)$ be a most permissive responsive controller of a k-responsively controllable service S where each state $q \in Q$ is equipped with the set $\psi^*(q)$ of valid responsive choices at state q.

Then, a compact canonical k-responsive controller $\widehat{\mathcal{C}}_{rp,k}(S)$ of service S is the service constructed from $mp_{rp,k}(S)^{\psi^*(q)}$ as defined by Definition 3.30.

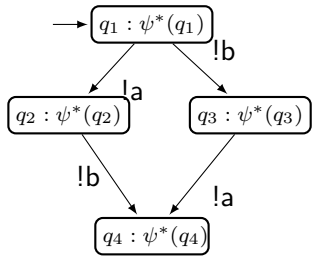
The constructed service $\widehat{\mathcal{C}}_{rp,k}(S)$ is a k-responsive controller of service S.

Corollary 4.9 ($\widehat{\mathcal{C}}_{rp,k}(S)$ is a k-responsive controller of S).
For each message bound $k \in \mathbb{N}$ and each k-responsively controllable service S :

$$\widehat{\mathcal{C}}_{rp,k}(S) \in rp_k Controllers(S).$$

Proof. Let $[mp_{rp,k}(S), \varphi]$ be an annotated most permissive k-responsive controller of S, and $\mathcal{C}_{rp,k}(S)$ be constructed from $[mp_{rp,k}(S), \psi^*]$ as described in Definition 4.8. By construction of $\widehat{\mathcal{C}}_{rp,k}(S)$, we obtain $\widehat{\mathcal{C}}_{rp,k}(S) \in Comply_\gamma(mp_{rp,k}(S)^{\psi^*})$ (Lemma 3.33). Because $rp_k Controllers(S) = Comply_\gamma(mp_{rp,k}(S)^{\psi^*})$ (Lemma 3.69), then $\widehat{\mathcal{C}}_{rp,k}(S) \in rp_k Controllers(S)$ follows. □

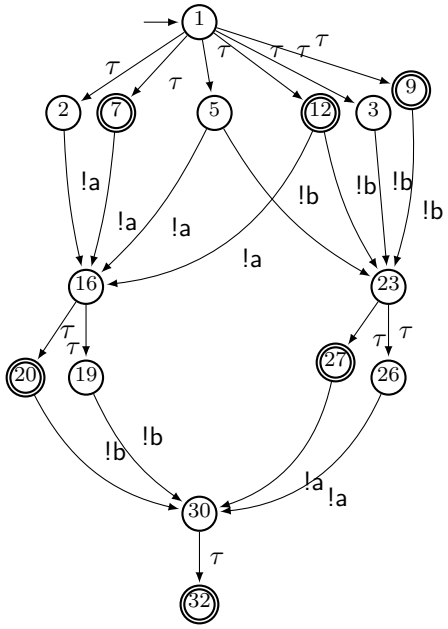
The following example shows two k-responsive controllers of a service automaton constructed from its responsive operating guideline.



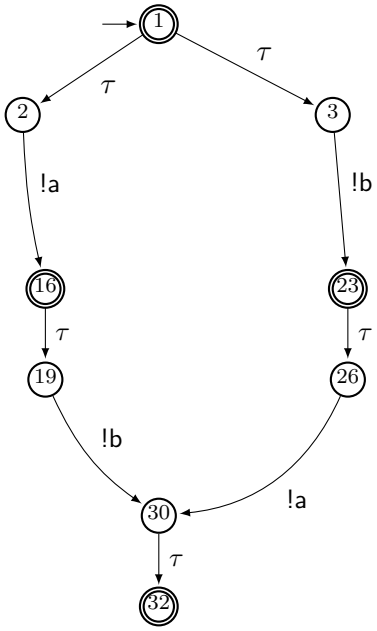
$$
\begin{aligned}
\psi^*(q_1) &= \{\ \{!a\},\{!b\},\\
&\quad \{!a,!b\},\{!a,\mathit{final}\},\{!a,\mathit{final}\},\{!b,\mathit{final}\},\\
&\quad \{!a,!b,\mathit{final}\},\ \},\\
\psi^*(q_2) &= \{\ \{!b\},\\
&\quad \{!b,\mathit{final}\}\ \}\\
\psi^*(q_2) &= \{\ \{!a\},\\
&\quad \{!a,\mathit{final}\}\ \}\\
\psi^*(q_4) &= \{\ \{\mathit{final}\}\ \}
\end{aligned}
$$

(a) $mp_{rp,k}(S_1)^{\psi^*}$

construct $\mathcal{C}_{rp,k}(S_1)$ ↙ ↘ construct $\widehat{\mathcal{C}}_{rp,k}(S_1)$



(b) $\mathcal{C}_{rp,k}(S_1)$



(c) $\widehat{\mathcal{C}}_{rp,k}(S_1)$

Figure 4.3.: A k-responsive operating guidelines $mp_{rp,k}(S_1)^{\psi^*}$ of S_1 (from Figure 4.1) and two canonical k-responsive controllers; $\mathcal{C}_{rp,k}(S_1)$ and $\widehat{\mathcal{C}}_{rp,k}(S_1)$ of S_1, both with $I = \{\ \}$ and $O = \{a, b\}$ for message bound $k = 1$.

Example 4.10. Consider $k = 1$ and service S_1 from Figure 4.1. Figure 4.3 illustrates on the top a k-responsive operating guideline $mp_{rp,k}(S_1)^{\psi^*}$ of S_1, the constructed $\mathcal{C}_{rp,k}(S_1)$ on the left hand side, and on the right hand side a k-responsive controller $\widehat{\mathcal{C}}_{rp,k}(S_1)$ of S_1 constructed from the operating guideline. The underlying structure $mp_{rp,k}(S_1)$ of the operating guideline has the same interface $I = \{\ \}$ and $O = \{a, b\}$ as $\mathcal{C}_{rp,k}(S_1)$ and $\widehat{\mathcal{C}}_{rp,k}(S_1)$ (omitted from the figure) for $k = 1$.

In comparison to $\mathcal{C}_{rp,k}(S_1)$ on its left hand side, the synthesized controller $\widehat{\mathcal{C}}_{rp,k}(S_1)$ is relatively smaller in size, as the construction procedure of $\widehat{\mathcal{C}}_{rp,k}(S_1)$ replaces each state q of the operating guideline with a fragment of non-deterministic τ events to only canonical choices of at each state of the operating guideline, but does not add an τ transition to any another event that is not described by any canonical choice at q.

In this example, at each state q_1, q_2, and q_3 of the operating guideline, there is only one *final* event that is not described by the canonical choice at the respective state, and this *final* event which is modeled by a final state of the service automaton at state 1, 16, and 23, respectively. For state q_4 of the operating guideline, every possible event at state q_4 (that is *final*) is described by a canonical choice at q_4. ◁

4.1.3. Experimental Results

In this section, we present the experimental results of constructing two types of a canonical responsive controller, a complete canonical responsive controller and a compact canonical responsive controller of various services.

Illustrated in Table 4.1 are six service models. The three services A:*Identity Card issue*, B:*Car Analysis*, and C:*Product Delivery* are industrial service models provided by a company. Service D:*Purchase Order* is taken from the *WS-BPEL* specification [Alves et al., 2007]. Service F:*Product Reservation* is taken from our own library. These models were specified in the service description language *WS-BPEL*. Then we translated the models into specifications into *open nets* using the compiler *BPEL2oWFN* [Lohmann, 2007] (See also Figure 2.9). We also experimented with an open net model of the simple mail transfer protocol E:*SMTP* [Klensin, 2001].

With the open net models of these services, we employed the tool *Wendy* [Lohmann and Weinberg, 2010] to construct their operation guidelines OG. Then we employed our tool *Maxis* [Parnjai, 2011c] to construct service automaton models of both complete canonical controller $\mathcal{C}_{rp,k}$ and compact canonical controller $\widehat{\mathcal{C}}_{rp,k}$ from an operating guideline.

Table 4.1.: Service S and its operating guideline $OG(S)$ for message bound $k = 1$

	Service S					$OG(S)$	
	$\lvert Q_S \rvert$	$\lvert \rightarrow_S \rvert$	$\lvert \stackrel{\tau}{\rightarrow}_S \rvert$	$\lvert I_S \rvert$	$\lvert O_S \rvert$	$\lvert Q \rvert$	$\lvert \rightarrow \rvert$
A : Identity Card Issue	14,569	71,332	66,500	2	9	1,537	15,115
B : Car Analysis	11,381	39,865	27,231	6	9	1,449	13,863
C : Product Delivery	4,148	13,832	9,288	5	9	1,377	13,838
D : Purchase Order	402	955	675	4	6	169	1,182
E : SMTP	60	92	0	9	5	470	3,051
F : Product Reservation	28	33	16	2	8	370	3,083

Though the tool *Wendy* requires an input as an open net model to produce its operating guidelines, it can also synthesize a service automaton model from an open net model. As our tool *Maxis* will generate a service automaton model as an output, we illustrate all services in Table 4.1 as service automata in order to compare their sizes on the same

Table 4.2.: Comparison of two canonical controllers, $\mathcal{C}_{rp,k}(S)$ and $\widehat{\mathcal{C}}_{rp,k}(S)$, for message bound $k = 1$

S	$\mathcal{C}_{rp,k}(S)$				$\widehat{\mathcal{C}}_{rp,k}(S)$			
	$\lvert Q\rvert$	$\lvert\Omega\rvert$	$\lvert\rightarrow\rvert$	$\lvert\overset{\tau}{\rightarrow}\rvert$	$\lvert Q\rvert$	$\lvert\Omega\rvert$	$\lvert\rightarrow\rvert$	$\lvert\overset{\tau}{\rightarrow}\rvert$
A	2,928,883	1,463,725	18,007,233	2,927,378	9,488	2	23,067	7,952
B	1,710,529	855,749	10,486,713	1,709,443	10,247	13	31,552	8,799
C	2,540,757	1,270,044	16,086,124	2,539,769	7,684	3	20,274	6,308
D	43,830	21,871	218,362	43,679	731	2	1,745	563
E	148,592	74,111	852,424	148,191	1,946	2	4,527	1,476
F	142,245	71,456	756,380	142,041	2,317	17	6,197	1,948

scale. To construct an operating guidelines from an output of *Maxis*, if needed, we can translate the output of *Maxis* into open net model using the tool *PnAPI* [Mennicke et al., 2009] (See also Figure 2.9).

Table 4.1 and Table 4.2 show the size of each service as a service automaton model. The size of a service is described in terms of the number of all states $|Q|$, the number of all final states $|\Omega|$, the number of all transitions $|\rightarrow|$, the number of τ transitions $|\overset{\tau}{\rightarrow}|$, the size of its input interface $|I|$, and the size of its output interface $|O|$. The size of the respective operating guideline $OG(S)$ is judged by the size of its underlying most permissive controller; the number of its states $|Q|$ and the number of its transitions $|\rightarrow|$.

As expected from its construction algorithm (Definition 3.23), the size of a complete canonical responsive controller $\mathcal{C}_{rp,k}$ as illustrated in Table 4.2 is growing exponentially in comparison to the size of its input operating guidelines. In comparison to the compact canonical controller (its construction algorithm is defined by Definition 3.30); however, the size of compact canonical responsive controller $\widehat{\mathcal{C}}_{rp,k}$ is relatively much smaller and is not growing in the same order as complete canonical responsive controller $\mathcal{C}_{rp,k}$.

4.2. Traces and Controllers

In this section, we study *trace* semantics for service automata, as lifted from the *traces* semantics which has been defined (e. g., in Hoare [1978], Roscoe [1998]) for labeled transition systems. We show that trace refinement of a canonical controller of a given service is necessary but not sufficient to decide its controller.

4.2.1. Trace Semantics

A service automaton S can be represented by the set of all its *traces*. A *trace* of service automaton S is a finite or infinite sequence of all events in the set Σ of communicating events of S. Recall from Section 2.1 that the set Σ is partitioned into the set $!\Sigma$ of message sending events and the set $?\Sigma$ of message receiving events (i. e., $\Sigma = !\Sigma \cup ?\Sigma$ and $!\Sigma \cap ?\Sigma = \emptyset$).

We define the *Trace* semantics for a service automaton as follows.

Definition 4.11 (Traces).
A *trace* of a service automaton $S = [Q, q_0, I, O, \rightarrow, \Omega]$ is a finite or infinite sequence of communicating message events in Σ from the initial state q_0 of S. The set of all *traces* of S is denoted by $\mathit{traces}(S)$.

Each trace σ of S is a word over Σ (i. e., $\sigma \in \Sigma$), this means, σ includes neither a non-communicating τ event nor a terminating *final* event. For the set Σ^* of all words over the set Σ of S, we have $\mathit{traces}(S) \subseteq \Sigma^*$.

The set $\mathit{traces}(S)$ of all traces of a given service S is never an empty set, as it always contains an empty trace ϵ and is prefix closed as $\sigma.m \in \mathit{traces}(S)$ implies $\sigma \in \mathit{traces}(S)$.

Property 4.12 For each service automaton S, the set $\mathit{traces}(S)$ of all traces of S is *non-empty* and *prefix closed*.

By definition, a trace σ of a service automaton S is *maximal* whenever for each state q_n with $q_0 \stackrel{\sigma}{\Rightarrow} q_n$, none of the communicating message events $e \in \Sigma$ is enabled at state q_n, i. e., $q_n \stackrel{e}{\not\rightarrow}$. Nevertheless, a maximal trace σ of service S does not indicate that S can terminate successfully with a final state. This is because service S may either terminate with a deadlock state after having performed σ, or diverge, i. e., perform nothing but infinitely many steps of non-communicating τ events.

4.2.2. Trace Refinement and Equivalence

Next, we define trace refinement relations for service automata. We give the definition of trace preorder $\sqsubseteq_{tr}$ as follows.

Definition 4.13 (Trace refinement).
Let R and S be two interface equivalent service automata. Then, service R *refines* service S *under traces*, denoted by $S \sqsubseteq_{tr} R$, iff every trace of S is also a trace of R, i. e., $\mathit{traces}(S) \supseteq \mathit{traces}(R)$. Then, the set of all services that refines service S under traces is denoted by

$$\mathit{tr\text{-}refine}(S) = \{\ R \mid S\ \sqsubseteq_{tr}\ R\ \}.$$

The $\sqsubseteq_{tr}$ refinement relation is a *preorder* relation, as it is a reflexive and transitive relation. We also use the notion trace preorder for trace refinement relation throughout the thesis.

The trace refinement relation $\sqsubseteq_{tr}$ induces an equivalence relation $=_{tr}$ that relates service automata with identical set of traces.

Definition 4.14 (Trace equivalence).
Two interface equivalent service automata R and S are *trace equivalent*, denoted by $S =_{tr} R$, iff they have exactly the same set of traces, i. e., $traces(R) = traces(S)$. Then, the set of all services that are equivalent to service S under traces is denoted by

$$tr\text{-}equiv(S) \;=\; \{\; R \mid R \;=_{tr}\; S \;\}.$$

The $=_{tr}$ is an *equivalence* relation, as it is a reflexive, transitive, and symmetric relation.

4.2.3. Relationship with Deadlock Freedom and Responsiveness

As trace semantics cannot distinguish between deadlock and successful termination, it is straightforward to see that the traces preorder relation is weaker than our deadlock freedom preorder and responsiveness preorder. In this section, we illustrate the relationship between trace refinement and the set of deadlock-free controllers and the set of responsive controllers of a given service.

Relationship with Deadlock Freedom

For the set of all k-deadlock-free controllers of service S, there exists at least three elements in the set that are in the traces preorder *larger than* or *equal to* any other k-deadlock-free controller S in the set. One element is a most permissive k-deadlock-free controller of S (Definition 3.40) and the other two elements are the two canonical k-deadlock-free controllers of S introduced in Section 4.1.1.

With the following property, we show that a most permissive k-deadlock-free controller of service S and a canonical k-deadlock-free controller of service S are trace equivalent by construction.

Property 4.15 For each message bound $k \in \mathbb{N}$ and each k-deadlock-freely controllable service S:

$$mp_{df,k}(S) \;=_{tr}\; \mathcal{C}_{df,k}(S) \;=_{tr}\; \widehat{\mathcal{C}}_{df,k}(S).$$

The following corollary illustrates that every k-deadlock-free controller P of service S refines a canonical k-deadlock-free controller under traces, for each $k \in \mathbb{N}$.

Corollary 4.16 (Trace refinement of canonical deadlock-free controller).
For each message bound $k \in \mathbb{N}$ and each two interface compatible services S and P :

$$P \in df_k Controllers(S) \;\Rightarrow\; P \in tr\text{-}refine(mp_{df,k}(S)).$$

Proof. Consider a k-deadlock-freely controllable service S and its most permissive k-deadlock-free controller $mp_{df,k}(S)$. Suppose $P \in df_k Controllers(S)$. This means that the

controller $mp_{df,k}(S)$ strongly simulates P follows (Lemma 3.51). As strong simulation preorder implies trace preorder, $mp_{df,k}(S) \sqsubseteq_{tr} P$ follows. □

Nevertheless, the reverse of Corollary 4.16 does not hold. This means that trace refinement of a most permissive deadlock-free controller of a given service is not sufficient to decide its deadlock-free controller. As for a given service, its most permissive deadlock-free controller and its canonical deadlock-free controller are trace equivalent, a similar argument to Corollary 4.16 is also applicable for its canonical deadlock-free controllers.

The following example illustrates four services automata, each of which refines a canonical k-deadlock-free controller of S_1 from Figure 4.2 under traces.

Example 4.17. Figure 4.4 illustrates four service automata; P_1, P_2, P_3, and P_4, each of which has the same interface ($I = \{\}$ and $O = \{a, b\}$). For readability purpose, the interface in each sub-figure is omitted. Clearly, the interface of each service is the same as the interface of the two canonical k-deadlock-free controllers, $\mathcal{C}_{df,k}(S_1)$ and $\widehat{\mathcal{C}}_{df,k}(S_1)$, of S_1 for $k = 1$ from Figure 4.2, and the interface of each service is compatible with that of each service from Figure 4.1.

Consider $k = 1$. Every service in Figure 4.4 is k-deadlock-freely controllable and strongly complies with the k-deadlock-free operating guideline $mp_{df,k}(S_1)$ in Figure 4.2. This means that they are all k-deadlock-free controllers of services S_1 (and also of S_2 as S_1 and S_2 are equivalent under k-deadlock-freedom). Therefore, each service in Figure 4.4 refines each canonical k-deadlock-free controller of S_1 in Figure 4.2 under traces. ◁

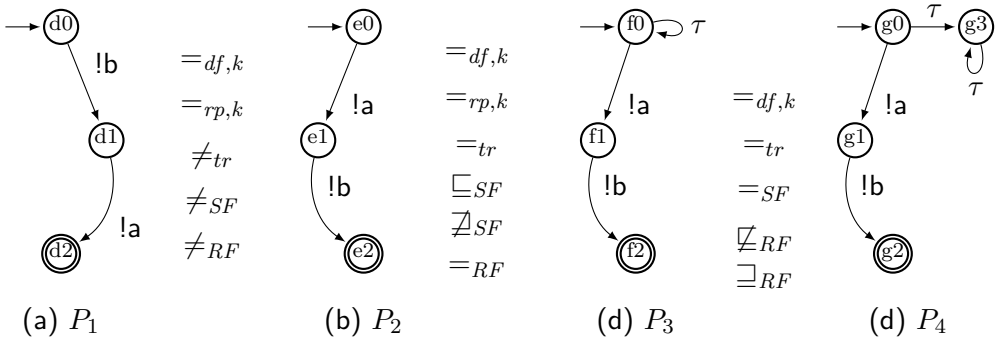


Figure 4.4.: Services P_1, P_2, P_3, and P_4, each of which has interface $I = \{\}$ and $O = \{a, b\}$. For message bound $k = 1$ on each message channel.

Relationship with Responsiveness

With the following property, we show that a most permissive k-responsive controller and a canonical k-responsive controller of service S are trace equivalent by construction.

Property 4.18 For each message bound $k \in \mathbb{N}$ and each k-responsively controllable service S :

$$mp_{rp,k}(S) \ =_{tr} \ \mathcal{C}_{rp,k}(S) \ =_{tr} \ \widehat{\mathcal{C}}_{rp,k}(S).$$

Similar to deadlock freedom preorder, every k-responsive controller P of service S refines a canonical k-responsive controller under traces for each message bound $k \in \mathbb{N}$.

Corollary 4.19 (Trace refinement of canonical responsive controller). For each message bound $k \in \mathbb{N}$ and each two interface compatible services S and P :

$$P \in rp_k Controllers(S) \ \Rightarrow \ P \in tr\text{-}refine(mp_{rp,k}(S)).$$

Proof. Consider a most permissive k-responsive controller $mp_{rp,k}(S)$ of a k-responsively controllable service S. Suppose $P \in rp_k Controllers(S)$. This means that $mp_{rp,k}(S)$ structurally simulates P follows (Lemma 3.69). As structural simulation preorder implies trace preorder, then $mp_{rp,k}(S) \sqsubseteq_{tr} P$ follows. □

Nevertheless, the reverse of Corollary 4.19 does not hold. Similarly to deadlock freedom, trace refinement of a most permissive responsive controller of a given service is not sufficient to decide its responsive controller. As for a given service, its most permissive responsive controller and its canonical responsive controller are trace equivalent, a similar argument to Corollary 4.19 is also applicable for its canonical responsive controllers.

We illustrate this phenomenon with the following example.

Example 4.20. Consider message bound $k = 1$ and the four service automata, P_1, P_2, P_3, and P_4 from Figure 4.4, each of which has the same interface ($I = \{\}$ and $O = \{a, b\}$). For readability purpose, the interface in each sub-figure is omitted.

Every service except service P_4 is k-responsively controllable. Each of P_1, P_2, and P_3 structurally complies with the k-responsive operating guideline of S_1 in Figure 4.3, but P_4 does not comply due to state h_3. This means that P_1, P_2, and P_3 are all k-responsive controllers of service S_1 (and also of S_2 as S_1, S_2 and S_3 are all equivalent under k-responsiveness), but P_4 is not a k-responsive controller of S_1. Nevertheless, we see that every service in Figure 4.4 refines the canonical k-deadlock-free controller $\mathcal{C}_{rp,k}(S_1)$ in Figure 4.3 under traces. ◁

4.3. Stable Failures and Deadlock-free Controllers

In the previous section, we have introduced trace semantics for service automata and have shown that a service automaton S can be represented by the set of all its *traces*. Nevertheless, trace semantics does not distinguish between deadlock and successful termination, as trace semantics describes what a service *can* do, but not about what it *cannot* do or what it *fails to* do.

In this section, we introduce *stable failures* semantics for service automata, lifted from the *stable failures* semantics which has been defined (e. g., in Roscoe [1998], Brookes et al. [1984], Hoare [1978, 1985b]) for labeled transition systems. For this purpose, we take into account a successfully terminating event *final* when calculating the set of events that a service fails to do.

We illustrate that there is no immediate coincidence between our deadlock freedom preorder (and equivalence) and the stable failures preorder (and equivalence). Nevertheless, the stable failures refinement of a canonical deadlock-free controller of a given service is sufficient to decide its deadlock-free controller.

4.3.1. Stable Failures Semantics

A *stable failure* of a service automaton is a pair of a *trace* σ and a *refusal set* X of events that are not enabled at a *stable state* of the service automaton after having performed the trace σ. Recall from Definition 2.4, Section 2.1 that a *stable state* is a state that does not enable an internal τ event, that is, $\tau \notin \mathit{enable}(q)$.

We define the *stable failures* semantics for a service automaton as follows.

Definition 4.21 (Stable failures).
Let q be a stable state of a service automaton $S = [Q, q_0, I, O, \rightarrow, \Omega]$. A *stable failure* of a service automaton S is a pair $[\sigma, X]$ that consists of

1. a trace σ of communicating message events from the initial state of S to state q, i.e. $q_0 \stackrel{\sigma}{\Rightarrow}_S q$ and $\sigma \subseteq \Sigma^*$; and
2. a stable refusal set X at stable state q that is defined as $X = \mathit{Refuse}(q) = \Sigma \cup \{\mathit{final}\} \backslash \mathit{enable}(q)$.

The set of all stable failures of S is denoted by $\mathit{failures}(S)$.

A *stable failure* of a service automaton S is a pair $[\sigma, X]$. With σ, we denote a (possibly empty) sequence σ of events, i. e., there is a sequence σ of events from the initial state q_{0S} to state q in S. With the set $X = \mathit{Refuse}(q)$, we denote the set of events that contains either a communicating event $e \in X$ that is refused at stable state q after having executed σ (i. e., $q \stackrel{e}{\not\rightarrow}$), or $e = \mathit{final}$ indicating that S does not yet reach a final state at q after having executed σ (i. e., $q \notin \Omega$).

It is important to notice that a trace σ of a given service S is a finite or infinite sequence of communicating events (i. e., $\sigma \subseteq \Sigma^*$). Nevertheless, the stable refusal set $\mathit{Refuse}(q)$ at stable state q after having performed σ also takes into account a special terminating event *final* (i. e., $X = \mathit{Refuse}(q) = \Sigma \cup \{\mathit{final}\} \setminus \mathit{enable}(q)$).

Property 4.22 illustrates that every trace σ of service S is associated with a failure $[\sigma, X]$ of service S.

Property 4.22 For each service S:

$$\sigma \in \mathit{traces}(S) \;\Rightarrow\; [\sigma, X] \in \mathit{failures}(S).$$

Property 4.23 illustrates that if a service can refuse the set X of events after performing σ, then a service can also refuse any subset Y of X after performing σ.

Property 4.23 For each service S:

$$[\sigma, X] \in \mathit{failures}(S) \;\wedge\; Y \subseteq X \;\Rightarrow\; [\sigma, Y] \in \mathit{failures}(S).$$

In case σ is a trace that a given service S can perform, certainly either service S reaches a stable state, or reaches a non-stable state that is a responsive state, or diverges after having performed σ. If service S reaches a state that is non-stable and responsive, then at this point S can perform infinitely many steps of non-communicating τ events and is also able to perform a communicating event after having performed σ. If service S reaches a divergent state after having perform a trace σ, then at this point it is no longer possible for service S to reach a stable state. In both cases, $(\sigma, \{\}) \in \mathit{failures}(S)$ also holds due to Property 4.22 and 4.23.

Property 4.24 illustrates that if a service cannot perform a trace that terminates with a particular event, then such an event must have been refused by this service at some previous state.

Property 4.24 For each service S:

$$\sigma.m \notin \mathit{traces}(S) \;\wedge\; m \in \Sigma \;\Rightarrow\; \forall [\sigma, X] \in \mathit{failures}(S) :: m \in X.$$

Property 4.25 illustrates that if a service can refuse the set X of events after performing σ, then a service can perform any other event that is not in X after performing σ.

Property 4.25 For each service S:

$$[\sigma, X] \in \mathit{failures}(S) \;\wedge\; m \in (\Sigma \setminus X) \;\Rightarrow\; \sigma.m \in \mathit{traces}(S).$$

A *final* state q of a service automaton S can be either a stable state or non-stable state. In case a final state q is stable, service S does not refuse a *final* event after having performing a trace σ and therefore S *can* terminate successfully with a final state after having performed σ. This is illustrated by Property 4.26.

Property 4.26 For each service S :

S can reach a final state after performing $\sigma \;\Leftrightarrow\; \exists [\sigma, X] \in \mathit{failures}(S) :: \mathit{final} \notin X.$

A *deadlock* state q is a stable and non-final state without an outgoing transition, that is, it refuses to perform every event from $\Sigma \cup \{final\}$, i. e., $Refuse(q) = \Sigma \cup \{final\}$. A service is deadlock-free whenever there is no failure that refuses every event from $\Sigma \cup \{final\}$. This is illustrated by Property 4.27.

Property 4.27 For each service S :

$$S \text{ is deadlock-free } \Leftrightarrow \forall [\sigma, X] \in failures(S) :: X \neq \Sigma \cup \{final\}.$$

We illustrate the set of all stable failures of a service automaton with the following examples.

Example 4.28. Consider services P_1, P_2, P_3, and P_4 from Figure 4.4. They all have the same set of interface ($I = \{\}$ and $O = \{a, b\}$) and the same set of action events ($\Sigma = \{!a, !b\}$). Their stable failures are illustrated as follows.

$$failures(P_1) = \{\ [\epsilon, \{!a, final\}], [!b, \{!b, final\}], [!b!a, \{!a, !b\}]\ \}.$$
$$failures(P_2) = \{\ [\epsilon, \{!b, final\}], [!a, \{!a, final\}], [!a!b, \{!a, !b\}]\ \}.$$
$$failures(P_3) = \{\ [\epsilon, \{\ \}], [!a, \{!a, final\}], [!a!b, \{!a, !b\}]\ \}.$$
$$failures(P_4) = \{\ [\epsilon, \{\ \}], [!a, \{!a, final\}], [!a!b, \{!a, !b\}]\ \}.$$

We see that services P_3 and P_4 have the same set of stable failures, each of which does not contain the stable failure $[\epsilon, \{!b, final\}]$. This phenomenon is due to $\tau \in enable(f0)$ of P_3 as well as $\tau \in enable(g)$ and $\tau \in enable(g3)$ of P_4, as states $f0$, $g0$, and $g3$ are not stable. ◁

4.3.2. Stable Failures Refinement and Equivalence

A service automaton R *refines* service automaton S *under stable failures*, denoted by $S \sqsubseteq_{SF} R$ whenever every failure of R is also a failure of S.

Definition 4.29 (Stable failures refinement).
A service automaton Q *refines* service automaton P *under stable failures*, denoted by $S \sqsubseteq_{SF} R$, iff $failures(S) \supseteq failures(R)$. Then, the set of all services that refine service S under stable failures is denoted by

$$f\text{-}refine(S) \ = \ \{\ R \mid S \ \sqsubseteq_{SF} R\ \}.$$

The $\sqsubseteq_{SF}$ refinement relation is a *preorder* relation, as it is a reflexive and transitive relation. We also use the notion stable failures preorder for a stable failures refinement relation throughout the thesis.

Intuitively, $S \sqsubseteq_{SF} R$ means that every trace σ of R can be replayed by S and every refusal set in R after having performed σ can also be refused by S after having performed

the same trace σ. This means that service R can neither accept nor refuse an event unless service S does.

The stable failures refinement $\sqsubseteq_{SF}$ induces an equivalence relation $=_{SF}$ that relates services with identical sets of failures.

Definition 4.30 (Stable failures equivalence).
Two service automata S and R are *equivalent under stable failures*, denoted by $S =_{SF} R$, iff they have exactly the same set of stable failures, i. e., $\mathit{failures}(R) = \mathit{failures}(S)$. Then, the set of all services that are equivalent to service S under stable failures is denoted by

$$\mathit{f\text{-}equiv}(S) = \{ R \mid S =_{SF} R \}.$$

With the following lemma, we show that a (complete) canonical deadlock-free controller $\mathcal{C}_{df,k}(S)$ and a compact canonical deadlock-free controller $\widehat{\mathcal{C}}_{df,k}(S)$ of service S, as introduced in Section 4.1.1, by construction are equivalent under stable failures.

Lemma 4.31 (Canonical deadlock-free controllers are equivalent under stable failures).
For each message bound $k \in \mathbb{N}$ and each k-deadlock-freely controllable service S:

$$\mathcal{C}_{df,k}(S) =_{SF} \widehat{\mathcal{C}}_{df,k}(S).$$

Proof. Let $\mathcal{C}_{df,k}(S)$ and $\widehat{\mathcal{C}}_{df,k}(S)$ be constructed from a k-deadlock-free operating guideline $\mathit{mp}_{df,k}(S)^{\varphi}$ by Definition 4.1 and Definition 4.3 respectively. As the three controllers; $\mathcal{C}_{df,k}(S)$, $\widehat{\mathcal{C}}_{df,k}(S)$, and $\mathit{mp}_{df,k}(S)$, have the same interface, they also have the same set of communicating events. Let Σ be the set of all communicating events of each of the three. Because $\mathcal{C}_{df,k}(S)$ and $\widehat{\mathcal{C}}_{df,k}(S)$ are both deadlock-free controllers of S (Corollary 4.2 and Corollary 4.4 respectively), $\mathcal{C}_{df,k}(S), \widehat{\mathcal{C}}_{df,k}(S) \in \mathit{Comply}_{\beta}(\mathit{mp}_{df,k}(S)^{\varphi})$ holds (Lemma 3.51).

$\sqsubseteq_{SF}$: Suppose $[\sigma, X] \in \mathit{failures}(\widehat{\mathcal{C}}_{df,k}(S))$ and that there exists $\widehat{q_0} \stackrel{\sigma}{\Rightarrow} \widehat{q}$ with $X = \mathit{Refuse}(\widehat{q})$ in $\widehat{\mathcal{C}}_{df,k}(S)$. If $\widehat{q}$ is a stable state, i. e., $\tau \notin \mathit{enable}(\widehat{q})$, then by definition $\mathit{enable}(\widehat{q}) = (\Sigma \cup \{\mathit{final}\}) \setminus X$ follows. Otherwise $\tau \in \mathit{enable}(\widehat{q})$ holds, so that by definition $X = \{\}$ follows.

Consider state q_m in $\mathit{mp}_{df,k}(S)$ and a strong simulation $\widehat{\varrho}$ with $[\widehat{q}, q_m] \in \widehat{\varrho}$. By construction of $\widehat{\mathcal{C}}_{df,k}(S)$, there exists $ch \in \varphi(q_m)$ with $ch = \mathit{enable}(\widehat{q})$.

Because $\mathcal{C}_{df,k}(S)$ is constructed from $\mathit{mp}_{df,k}(S)^{\varphi}$ by replacing q_m with a fragment of non-deterministic τ events of all choices at $\varphi(q_m)$, this means, there exists also a state q in $\mathcal{C}_{df,k}(S)$ and a strong simulation ϱ with $[q, q_m] \in \varrho$ such that $ch = \mathit{enable}(q)$. It follows that either (1) q is not stable in $\mathcal{C}_{df,k}(S)$ and $X = \{\}$ or (2) q is stable in $\mathcal{C}_{df,k}(S)$ and $\mathit{Refuse}(q) = X$.

For both cases, we conclude that $[\sigma, X] \in \mathit{failures}(\mathcal{C}_{df,k}(S))$ also holds and by definition $\mathcal{C}_{df,k}(S) \sqsubseteq_{SF} \widehat{\mathcal{C}}_{df,k}(S)$ follows.

$\sqsupseteq_{SF}$: Suppose $[\sigma, X] \in \mathit{failures}(\mathcal{C}_{df,k}(S))$ and there exists $q_0 \stackrel{\sigma}{\Rightarrow} q$ with $X = \mathit{Refuse}(q)$ in $\mathcal{C}_{df,k}(S)$.

Consider state q_m in $mp_{df,k}(S)$ and a strong simulation ϱ with $[q, q_m] \in \varrho$. By construction of $\mathcal{C}_{df,k}(S)$, there exists $ch \in \varphi(q_m)$ with $ch = \mathit{enable}(q)$. If q is a stable state, i. e., $\tau \notin \mathit{enable}(q)$ holds, then by definition $X = (\Sigma \cup \{\mathit{final}\}) \setminus \mathit{enable}(q)$ follows. Otherwise $\tau \in \mathit{enable}(q)$ holds, and by definition $X = \{\}$ follows.

In case ch is a canonical choice, this means that there exist $ch' \in \varphi(q_m)$ with $ch' \neq ch$ such that either $ch \subsetneq ch'$ or $ch \cap ch' = \emptyset$ holds. It follows that there must be a state $\widehat{q}$ in $\widehat{\mathcal{C}}_{df,k}(S)$ and a strong simulation $\widehat{\varrho}$ with $[\widehat{q}, q_m] \in \widehat{\varrho}$ such that $ch = \mathit{enable}(\widehat{q})$. It follows that $\widehat{q}$ is stable in $\widehat{\mathcal{C}}_{df,k}(S)$ and $\mathit{Refuse}(q) = X$.

In case ch is not a canonical choice, this means, there must be a canonical choice $ch' \in \varphi(q_m)$ with $ch' \subsetneq ch$. By construction of $\widehat{\mathcal{C}}_{df,k}(S)$, there exists a state $\widehat{q'}$ in $\widehat{\mathcal{C}}_{df,k}(S)$ with $(\widehat{q'}, q_m) \in \widehat{\varrho}$ such that $ch' = \mathit{enable}(\widehat{q'})$. Because $ch' \subsetneq ch$, it follows that $\mathit{enable}(\widehat{q'}) \subsetneq \mathit{enable}(\widehat{q})$. It follows that either (1) $\widehat{q'}$ is not stable in $\widehat{\mathcal{C}}_{df,k}(S)$ and $X = \{\}$ or (2) $\widehat{q'}$ is stable in $\widehat{\mathcal{C}}_{df,k}(S)$ and $\mathit{Refuse}(q') \supseteq \mathit{Refuse}(q)$. If $\widehat{\mathcal{C}}_{df,k}(S)$ can refuse the set $\mathit{Refuse}(q')$ after performing σ, then $\widehat{\mathcal{C}}_{df,k}(S)$ can also refuse any subset of $\mathit{Refuse}(q')$ including an empty set after performing σ (Property 4.23).

For all cases, we conclude that $[\sigma, X] \in \mathit{failures}(\widehat{\mathcal{C}}_{df,k}(S))$ also holds and by definition $\mathcal{C}_{df,k}(S) \sqsupseteq_{SF} \widehat{\mathcal{C}}_{df,k}(S)$ follows.

Thus, the lemma holds. □

Example 4.32. Consider the most permissive k-deadlock-free controller $mp_{df,k}(S_1)$ and the two canonical k-deadlock-free controllers from Figure 4.2; $\mathcal{C}_{df,k}(S_1)$ and $\widehat{\mathcal{C}}_{df,k}(S_1)$. All are deadlock-free controllers of service S_1 from Figure 4.4. Their stable failures are illustrated as follows.

$$\begin{aligned}
\mathit{failures}(mp_{df,k}(S_1)) = \{\ & [\epsilon, \{\mathit{final}\}], \\
& [!b, \{!b, \mathit{final}\}], [!b!a, \{!a, !b\}], [!a, \{!a, \mathit{final}\}], [!a!b, \{!a, !b\}]\ \}. \\
\mathit{failures}(\mathcal{C}_{df,k}(S_1)) = \{\ & [\epsilon, \{!a, \mathit{final}\}], [\epsilon, \{!b, \mathit{final}\}], \\
& [!b, \{!b, \mathit{final}\}], [!b!a, \{!a, !b\}), [!a, \{!a, \mathit{final}\}], [!a!b, \{!a, !b\}]\ \}. \\
\mathit{failures}(\widehat{\mathcal{C}}_{df,k}(S_1)) = \mathit{failures}(\mathcal{C}_{df,k}(S_1)).
\end{aligned}$$

The three controllers have the same interface; $\mathcal{C}_{df,k}(S_1)$ and $\widehat{\mathcal{C}}_{df,k}(S_1)$ have the same set of stable failures, this means, $\mathcal{C}_{df,k}(S_1) =_{SF} \widehat{\mathcal{C}}_{df,k}(S_1)$, whereas every stable failure of $mp_{df,k}(S_1)$ is included in a stable failure of both $\mathcal{C}_{df,k}(S_1)$ and $\widehat{\mathcal{C}}_{df,k}(S_1)$, this means that $mp_{df,k}(S_1)$ refines both $\mathcal{C}_{df,k}(S_1)$ and $\widehat{\mathcal{C}}_{df,k}(S_1)$ under stable failures.

In comparison to services P_1, P_2, P_3, and P_4 from Figure 4.4, we see from Example 4.28 that all stable failures of each of the four services P_1, P_2, P_3, and P_4 are included in the set of all stable failures of $\mathcal{C}_{rp,k}(S_1)$ (and of $\widehat{\mathcal{C}}_{rp,k}(S_1)$). Therefore, each of the four services P_1, P_2, P_3, and P_4 refines $\mathcal{C}_{df,k}(S_1)$ (and $\widehat{\mathcal{C}}_{df,k}(S_1)$) under stable failures. ◁

4.3.3. Relationship with Deadlock Freedom

In this section, we illustrate a relationship between stable failures and deadlock freedom.

For synchronously communicating services, Stahl [2009] has shown a result indicating a coincidence between stable failures refinement and deadlock freedom inclusion. Nevertheless, for asynchronously communicating services, there is no immediate coincidence between deadlock freedom inclusion with stable failures refinement. This phenomenon is due to the nature of a service that communicates asynchronously with its compatible partner via a bounded and unordered message buffer. Such a setting allows each service to perform either unblocking send message events or unblocking receive message events or internal events, as well as to consume a sequence of messages from the message buffer in different order it is produced by its partner.

The following example illustrates that there is no immediate coincidence between our deadlock freedom inclusion and stable failure refinement for asynchronously communicating services.

Example 4.33. Consider services P_1, P_2, P_3, and P_4 in Figure 4.4 for message bound $k = 1$ on each message channel. Service P_3 refines service P_2 under stable failures ($P_2 \sqsubseteq_{SF} P_3$), whereas service P_3 and P_4 are equivalent under stable failures ($P_3 =_{SF} P_4$). See also Example 4.28. Nevertheless, services P_1, P_2, P_3, and P_4 are equivalent under k-deadlock freedom ($P_2 =_{df,k} P_2 =_{df,k} P_3 =_{df,k} P_4$).

Consider services S_2 and S_3 in Figure 4.1 for $k = 1$. Service S_3 refines service S_2 under stable failures ($S_2 \sqsubseteq_{SF} S_3$, similarly to $P_2 \sqsubseteq_{SF} P_3$). Nevertheless, service S_3 is a substitute for service S_2 under k-deadlock freedom preorder ($S_3 \sqsubseteq_{df,k} S_2$), but service S_2 is not a substitute for service S_3 under k-deadlock freedom inclusion ($S_2 \not\sqsubseteq_{df,k} S_3$). $\lhd$

Though there is no immediate coincidence between stable failures refinement and deadlock freedom inclusion, we observe that stable failures refinement on a given service introduces more possibilities for a service for communicating with other interacting service, as the failure refined service refuses less events after having executed a trace. By doing so, it impose less communication restriction to the other interacting service.

Next, we prove that every pair of services that satisfies the stable failures refinement also satisfies the substitution deadlock freedom inclusion.

Lemma 4.34 (Stable failures refinement implies deadlock freedom inclusion). For each message bound $k \in \mathbb{N}$ and each two k-deadlock-freely controllable services S and R with equivalent interface :

$$R \in \textit{f-refine}(S)) \;\Rightarrow\; R \;\in\; df_k\textit{Inclusion}(S).$$

Proof. Suppose $R \in \textit{f-refine}(S)$, this means, $\textit{failures}(R) \subseteq \textit{failures}(S)$ by definition. Suppose $P \in df_k\textit{Controllers}(S)$. We will show that $P \in df_k\textit{Controllers}(R)$. As controller is a symmetric notion (Proposition 2.27), we will show that $R \in df_k\textit{Controllers}(P)$.

Let $mp_{df,k}(P)^{\varphi}$ be a k-deadlock-free operating guideline of P.

Because controller is a symmetric notion (Proposition 2.27), $S \in df_k Controllers(P)$ also holds. This means, S is simulated by $mp_{df,k}(P)$ and, for an arbitrary simulated pair $[q_S, q_m]$ of state, state q_S strongly corresponds to $\varphi(q_m)$ (Lemma 3.51). As strong simulation implies traces refinement, there exist (1) a trace σ_S in S with $q_{0S} \stackrel{\sigma_S}{\Rightarrow}_S q_S$ with $\sigma_S = \sigma_m$, and (2) a choice $ch \in \varphi(q_m)$ with $ch = enable(q_S)$.

Let $[\sigma_R, X_R] \in failures(R)$. Because $failures(R) \subseteq failures(S)$ holds by assumption, we have $traces(R) \subseteq traces(S)$ and $[\sigma_R, X_R] \in failures(S)$ immediately follows. Because $traces(R) \subseteq traces(S)$ holds, then $traces(R) \subseteq traces(mp_{df,k}(P))$ follows. Because $mp_{df,k}(S)$ is deterministic, then R is strongly simulated by $mp_{df,k}(P)$.

Let $[\sigma_S, X_S] \in failures(S)$ with $X_S = Refuse(q_S) = (\Sigma_S \cup \{final\}) \setminus enable(q_S)$.

Because $[\sigma_R, X_R] \in failures(R) \subseteq failures(S)$, then $\sigma_R = \sigma_S$ and $X_R \subseteq X_S$ follows.

Because S and R are interface equivalent by assumption, it follows from $X_R \subseteq X_S$ that $enable(q_R) \supseteq enable(q_S)$ must hold.

As $\sigma_R = \sigma_S$, the enabled events at q_R, that are in $enable(q_R)$ but not enabled at q_S, do not introduce an extra trace that is not described by S. Therefore, there must be $ch' \in \varphi(q_m)$ such that $ch' \supseteq ch$ and $ch' = enable(q_R)$.

This means, R is simulated by $mp_{df,k}(P)$ and, for a simulated pair $[q_R, q_m]$ of states, state q_R strongly corresponds to $\varphi(q_m)$. Thus, $R \in df_k Controllers(P)$ holds (Lemma 3.51) and $P \in df_k Controllers(R)$ follows (Proposition 2.27). That is, $df_k Controllers(S) \subseteq df_k Controllers(R)$ holds and $R \in df_k Inclusion(S)$ follows by definition. □

Nevertheless, the reverse of Lemma 4.34 does not hold. Lemma 4.34 suggests that stable failures refinement is finer than our deadlock freedom inclusion, as every service that refines service S under stable failures is a substitute for service S under k-deadlock freedom inclusion.

It is also important to notice that Lemma 4.34 holds only for services S and R that are k-deadlock-freely controllable.

Example 4.35. Consider services S_2 and S_3 in Figure 4.1 where service S_3 refines service S_2 under stable failures ($S_2 \sqsubseteq_{SF} S_3$). This means that service S_3 is a substitute for service S_2 under k-deadlock freedom ($S_3 \sqsubseteq_{df,k} S_2$) for message bound $k = 1$ on each message channel.

Similar for services P_2 and service P_3 in Figure 4.4: service P_3 refines service P_2 under stable failures ($P_2 \sqsubseteq_{SF} P_3$). This means that service P_3 is a substitute for service P_2 under k-deadlock freedom inclusion ($S_3 \sqsubseteq_{df,k} S_2$) for $k = 1$.

Given one service that is a substitute for another service under k-deadlock freedom inclusion. For instance, services P_2 is a substitute for service P_3 under k-deadlock freedom inclusion ($P_2 \sqsubseteq_{df,k} P_3$) for $k = 1$. It is not the case that P_2 refines P_3 under stable failures, however. ◁

As the stable failures refinement and deadlock freedom inclusion are both preorder relation induces the equivalence relation, we obtain the following corollary from Lemma 4.34.

Corollary 4.36 (Stable failures equivalence implies deadlock freedom equivalence).
For each message bound $k \in \mathbb{N}$ and each two interface compatible services S and R that are k-deadlock-freely controllable :

$$R \in \textit{f-equiv}(S) \;\Rightarrow\; R \;\in\; \textit{df}_k\textit{Equiv}_k(S).$$

Proof. As the preorder induces the equivalence relation, the proof of this corollary follows from Lemma 4.34. □

Nevertheless, the reverse of Lemma 4.34 and Corollary 4.36 do not hold. This means, the stable failures refinement (and equivalence) is indeed finer than the deadlock freedom inclusion (and equivalence).

Example 4.37. Consider services S_3 and S_4 in Figure 4.1. Service S_3 and S_4 are equivalent under stable failures ($S_3 =_{SF} S_4$). As suggested by Corollary 4.36, services S_3 and S_4 are also equivalent under k-deadlock freedom ($S_3 =_{df,k} S_4$) for message bound $k = 1$ on each message channel.

Similar for services S_3 and service S_4 in Figure 4.1: services S_3 and S_4 are equivalent under stable failures ($S_3 =_{SF} S_4$). Corollary 4.36 suggests that services S_3 and S_4 are also equivalent under k-deadlock freedom ($S_3 =_{df,k} S_4$) for $k = 1$.

Given two services that are equivalent under k-deadlock freedom, for instance, services P_2 and P_3 in Figure 4.4 with $P_2 =_{df,k} P_3$ for $k = 1$, it is not the case that P_2 and P_3 are equivalent under stable failures, however. ◁

Though the stable failures refinement and the deadlock freedom inclusion do not coincide, we illustrate that the stable failures refinement implies the k-deadlock freedom inclusion for any message bound $k \in \mathbb{N}$ for each message channel.

With distinguished properties of the canonical deadlock-free controller, we can establish a coincidence between the set of all deadlock-free controllers of service S and the set of all services that refines a canonical k-deadlock-free controller $\mathcal{C}_{df,k}(S)$ of S under stable failures. We illustrate this with Theorem 4.38.

Theorem 4.38 (Relationship of stable failures refinement and deadlock-free controllers).
For each message bound $k \in \mathbb{N}$ and each k-deadlock-freely controllable service S :

$$\textit{f-refine}(\mathcal{C}_{df,k}(S)) = \textit{df}_k\textit{Controllers}(S).$$

Proof. By construction, a canonical k-deadlock-free controller $\mathcal{C}_{df,k}(S)$ of S is derived from the k-deadlock-free operating guideline $mp_{df,k}(S)^{\varphi}$ of S and they are interface equivalent. We prove this lemma in two directions.

$\Rightarrow$: Suppose $P \in$ *f-refine*$(\mathcal{C}_{df,k}(S))$, that is, *failures*$(P) \subseteq$ *failures*$(\mathcal{C}_{df,k}(S))$ holds. Let $\mathcal{C}_{df,k}(S)$ be constructed from the operating guideline $mp_{df,k}(S)^{\varphi}$ according to Definition 4.1. We will show that P strongly complies with $mp_{df,k}(S)^{\varphi}$.

As *failures*$(P) \subseteq$ *failures*$(\mathcal{C}_{df,k}(S))$, every trace in P can be replayed by $\mathcal{C}_{df,k}(S)$. As *traces*$(mp_{df,k}(S)) =$ *traces*$(\mathcal{C}_{df,k}(S))$ holds (Property 4.15), every trace in P can also be replayed by $mp_{df,k}(S)$. As $mp_{df,k}(S)$ is a deterministic service automaton, it follows that there is a strong simulation relation between P and $mp_{df,k}(S)$ such that P is strongly simulated by $mp_{df,k}(S)$.

Let $[\sigma_P, X_P] \in$ *failures*$(P) \subseteq$ *failures*$(\mathcal{C}_{df,k}(S))$. Let q_P be a state in P with $q_{0P} \stackrel{\sigma_P}{\Rightarrow}_P q_P$. We consider a simulated pair $[q_P, q_m]$ of a state q_P in P that is strongly simulated by a state q_m in $mp_{df,k}(S)$.

- In case q_P is a stable state, we have $X_P =$ *Refuse*(q_P). As the refusal set is the complement set of the set of all enabled events by definition, it follows that $Y_P =$ *enable*$(q_P) = \Sigma_P \cup \{$*final*$\} \setminus$ *Refuse*(q_P).
  Because $[\sigma_P, X_P] \in$ *failures*$(\mathcal{C}_{df,k}(S))$ holds, there exists a stable state q_C in $\mathcal{C}_{df,k}(S)$ such that (1) q_C is simulated by state q_m, and (2) C refuses the same set X_P of events at state q_C as at state q_P and therefore C offers the same enable events Y_P at state q_C and state q_P.
  Because $\mathcal{C}_{df,k}(S)$ contains a nondeterministic internal alternatives of all choices from $\varphi(q_m)$ by construction, it follows that q_C strongly corresponds to $\varphi(q_m)$ and q_P also strongly corresponds to $\varphi(q_m)$.
- In case q_P is not a stable state reached in P after having performed σ_P, this means that we have $\tau \in$ *enable*(q_P). It follows that q_P also strongly corresponds to $\varphi(q_m)$ as $\{\tau\}$ is always a choice of $\varphi(q_m)$.

Therefore, P strongly complies with $mp_{df,k}(S)^{\varphi}$ and $P \in df_k$*Controllers*$(S))$ holds (Lemma 3.51).

$\Leftarrow$: Suppose $P \in df_k$*Controllers*(S), that is, P strongly complies with the operating guideline $mp_{df,k}(S)^{\varphi}$ of S (Lemma 3.51). We will show $P \in$ *f-refine*$(\mathcal{C}_{df,k}(S))$.

Consider $[\sigma, X] \in$ *failures*(P). We will show that $[\sigma, X] \in$ *failures*$(\mathcal{C}_{df,k}(S))$ holds.

Let $mp_{df,k}(S)^{\varphi}$ be an k-deadlock-free operating guideline of S.

As $\mathcal{C}_{df,k}(S)$ is a k-deadlock-free controller of S (Corollary 4.2) and $mp_{df,k}(S)$ is a most permissive k-deadlock-free controller of S (Proposition 3.43), then $mp_{df,k}(S)$ strongly simulates both $\mathcal{C}_{df,k}(S)$ and P (Proposition 3.44). By construction of $\mathcal{C}_{df,k}(S)$, *traces*$(mp_{df,k}(S)) =$ *traces*$(\mathcal{C}_{df,k}(S))$ holds (Property 4.15). This means that $\sigma \in$ *traces*$(mp_{df,k}(S))$ whenever $\sigma \in$ *traces*$(\mathcal{C}_{df,k}(S))$. Then, σ is also a trace of $\mathcal{C}_{df,k}(S)$.

As the refusal set of each state q is the complement set of the set of all enabled events at q by definition, that is, *Refuse*$(q) = \Sigma \cup \{$*final*$\} \setminus$ *enable*(q).

- Assume $X \neq \emptyset$ and $Y =$ *enable*$(q) = (\Sigma \cup \{$*final*$\}) \setminus X$. Because P is a k-deadlock-free controller of S by assumption, P strongly corresponds to $\varphi(q)$

(Lemma 3.51). Because $\mathcal{C}_{df,k}(S)$ contains a nondeterministic internal alternative of all choices from $\varphi(q)$ by construction. This means, there exists a stable state q_β in $\mathcal{C}_{df,k}(S)$ that offers exactly all enabled events described by Y. As X and Y are complement set of each other, then that $X = \mathit{Refuse}(q_\beta)$ is the refusal set at q_β.

– Assume $X = \emptyset$. This means that X is also a refusal set in $\mathcal{C}_{df,k}(S)$ as an empty set is a subset of every set.

Therefore, $[\sigma, X] \in \mathit{failures}(\mathcal{C}_{df,k}(S))$ holds and $\mathit{failures}(\mathcal{C}_{df,k}(S)) \supseteq \mathit{failures}(P)$ follow. Thus, $P \in \mathit{f\text{-}refine}(\mathcal{C}_{df,k}(S))$ follows by definition.

Thus, this theorem holds. □

Theorem 4.38 suggests that the set of all deadlock-free controllers of service S coincides with the set of all services that refine a canonical k-deadlock-free controller $\mathcal{C}_{df,k}(S)$ of S under stable failures.

Nevertheless, a canonical k-deadlock-free controller $\mathcal{C}_{df,k}(S)$ of S, as stated in Theorem 4.38, can be replaced only by an arbitrary k-deadlock-free controller of S that is equivalent to a canonical k-deadlock-free controller $\mathcal{C}_{df,k}(S)$ of S under stable failures, such as a compact canonical k-deadlock-free controller of S (cf. Lemma 4.31). This is due to the distinguished property of a canonical k-deadlock-free controller $\mathcal{C}_{df,k}(S)$ of S.

We illustrate the result from Theorem 4.38 with the following example.

Example 4.39. Consider services P_1, P_2, P_3, and P_4 in Figure 4.4 and message bound $k = 1$ on each message channel. We see that every service in the figure refines the canonical k-deadlock-free controller $\mathcal{C}_{df,k}(S_1)$ of S_1 illustrated in Figure 4.2. This means that every service is a k-deadlock-free controller of service S_1 illustrated in Figure 4.1 (also a k-deadlock-free controller of service S_2 as S_1 and S_2 are equivalent under k-deadlock freedom). ◁

Next, we prove in Lemma 4.40 that we can decide if service T is a substitute for service S under deadlock-free preorder by checking if a canonical k-deadlock-free controller $\mathcal{C}_{df,k}(S)$ of S refines a canonical k-deadlock-free controller $\mathcal{C}_{df,k}(T)$ of T under stable failures.

Lemma 4.40 (Stable failures refinement of canonical deadlock-free controllers).
For each message bound $k \in \mathbb{N}$ and each two k-deadlock-freely controllable services S and T :

$$T \sqsubseteq_{df,k} S \Leftrightarrow \mathcal{C}_{df,k}(T) \sqsubseteq_{SF} \mathcal{C}_{df,k}(S).$$

Proof. Given :

$$\begin{aligned} T \sqsubseteq_{df,k} S \;\Leftrightarrow\; & \{\text{by Definition 2.36}\} \\ & df_k Controllers(T) \supseteq df_k Controllers(T) \\ \Leftrightarrow\; & \{\text{by applying Thoerem 4.38 twice}\} \\ & \textit{f-refine}(\mathcal{C}_{df,k}(T)) \supseteq \textit{f-refine}(\mathcal{C}_{df,k}(S)) \\ \Leftrightarrow\; & \{\text{by Definition 4.29 and set implication}\} \\ & \mathcal{C}_{df,k}(T) \sqsubseteq_{SF} \mathcal{C}_{df,k}(S). \end{aligned}$$

Consequently, the lemma holds. □

With Corollary 4.41, we prove that we can decide if services S and T are deadlock-free equivalent by checking if two canonical deadlock-free controllers, $\mathcal{C}_{df,k}(S)$ of S and $\mathcal{C}_{df,k}(T)$ of T, are equivalent under stable failures.

Corollary 4.41 (Stable failures equivalence of canonical deadlock-free controllers).
For each message bound $k \in \mathbb{N}$ and each two k-deadlock-freely controllable services S and T :

$$T =_{df,k} S \;\Leftrightarrow\; \mathcal{C}_{df,k}(T) =_{SF} \mathcal{C}_{df,k}(S).$$

Proof. Follows from Lemma 4.40. □

With the aforementioned properties, a canonical k-deadlock-free controller $\mathcal{C}_{df,k}(S)$ of a given service S can be regarded as a *behavioral specification* of all k-deadlock-free controllers of service S for $k \in \mathbb{N}$. For checking whether an implementation (e. g., service P) *conforms* to the specification, we apply the stable failures refinement relation as the conformance relation. For service P that refines the specification $\mathcal{C}_{df,k}(S)$ under stable failures, we conclude that service P is indeed an implementation (of the specification $\mathcal{C}_{df,k}(S)$) of S, meaning P can interact deadlock-freely with S. We discuss the respective applications in Chapter 7.

4.4. Responsive Failures and Responsive Controllers

As the stable failures semantics considers a failure at a stable state, this means that stable failures semantics cannot distinguish a service that can refuse only the set X of actions from a service that can both refuse the set of actions and can *diverge*, i. e., performs nothing but a (possibly infinite) sequence of invisible τ-events (See the pair of services S_3 and S_4 from Figure 4.1 and Example 4.1, for instances).

In this section, we introduce the *responsive failures* semantics for service automata as an extension of *stable failures* semantics for service automata. The responsive failures

semantics distinguishes between deadlock and successfully termination, but does not distinguish between deadlock and divergent actions. We illustrate that the preorder induced by the responsive failures is stronger than the one induced by the stable failures. Finally, we establish the preorder induced by the responsive failures with the k-responsiveness preorder via both a canonical k-responsive controller and a canonical k-responsive service of a given service for each message bound $k \in \mathbb{N}$.

4.4.1. Responsive Failures Semantics

In this section, we propose the *responsive failures* semantics for service automata as an extension of *stable failures*.

One significant difference to *stable failures* is that responsive failures does not distinguish between deadlock and divergence. Technically, the stable failures consider a *stable refusal set* of events at every *stable* state, whereas the *responsive failures* consider the *responsive refusal set* of events at every *stable* τ-strongly connected component of a given service. Recall from Definition 2.9, Section 2.1 that a stable τ-strongly connected component is a strongly connected component in which every outgoing transition labeled with τ has reached a destination state within the same component.

We define a *responsive failure* semantics of a service automaton as follows.

Definition 4.42 (Responsive failure).
Let C be a stable τ-strongly connected component of a service automaton $S = [Q, q_0, I, O, \rightarrow, \Omega]$ and $q \in Q_C$ be a state contained in C where Q_C is the set of all states in C.

A *responsive failure* of a service automaton S is a pair $[\sigma, X]$ that consists of

1. a trace σ of external events from the initial state of S to state q, i.e. $q_0 \stackrel{\sigma}{\Rightarrow}_S q$ and $\sigma \subseteq \Sigma^*$; and
2. a responsive refusal set X at the stable τ-strongly connected component C that is defined as $X = \mathit{Refuse}(Q_C) = \Sigma \cup \{\mathit{final}\} \backslash \mathit{act}^*(Q_C)$.

The set of all responsive failures of S is denoted by $\mathit{rp\text{-}failures}(S)$.

A *responsive failure* of a service automaton S is a pair $[\sigma, X]$. With σ we denote a trace or a sequence of events of S, i.e. there is a (possibly empty) sequence σ of events from the initial state q_{0S} to a state within a stable τ-strongly connected component C. Clearly, every state containing the component C has the same trace from the initial state, as all states in the τ-strongly connected component can reached each another only by performing a non-communicating τ event.

For each stable τ-strongly connected component C, we calculate the set of all enabled non-invisible events at C as the set $\mathit{act}^*(Q_C)$ where Q_C is the set of all states in C. Recall from Definition 2.10 that the set $\mathit{act}^*(Q_C)$ is the union of all enabled communicating events of all states within the component C. Therefore, the *responsive refusal set* X is

the set of events that are not described by $act^*(Q_C)$, i. e., $X = (\Sigma \cup \{final\}) \setminus act^*(Q_C)$ where $\Sigma = !\Sigma \cup ?\Sigma$.

Similar to the stable failures semantics, it is important to notice that a trace σ of a given service S is a finite or infinite sequence of communicating events (i. e., $\sigma \subseteq \Sigma^*$). Nevertheless, the stable refusal set $Refuse(q)$ at stable state q after having performed σ also takes into account a special terminating event *final* (i. e., $X = Refuse(q) = \Sigma \cup \{final\} \setminus enable(q)$).

Property 4.43 illustrates that every trace σ of service S is associated with a failure $[\sigma, X]$ of service S.

Property 4.43 For each service S:

$$\sigma \in traces(S) \;\Rightarrow\; [\sigma, X] \in rp\text{-}failures(S).$$

Property 4.44 illustrates that if a service can responsively refuse the set X of events after performing σ, then a service can also responsively refuse any subset Y of X after performing σ.

Property 4.44 For each service S:

$$[\sigma, X] \in rp\text{-}failures(S) \;\wedge\; Y \subseteq X \;\Rightarrow\; [\sigma, Y] \in rp\text{-}failures(S).$$

In case σ is a trace that a given service S can perform, then certainly service S reaches a stable τ-strongly connected component C. If service S reaches either a divergent state or a deadlock state in C after having perform a trace σ, this means at this point it is no longer possible for service S to enable any event from $\Sigma \cup \{final\}$ state. In such cases, $act^*(Q_C) = \emptyset$ due to Lemma 2.12 and $[\sigma, \Sigma \cup \{final\}] \in rp\text{-}failures(S)$.

This means, responsive failures semantics treats a divergent state and a deadlock state similarly, as each of them responsively refuses every event in $\Sigma \cup \{final\}$.

Property 4.45 illustrates that if a service cannot perform a trace that terminates with particular event, then such event must be refuse by a service at some previous state.

Property 4.45 For each service S:

$$\sigma.m \notin traces(S) \;\wedge\; m \in I \cup O \;\Rightarrow\; \forall [\sigma, X] \in rp\text{-}failures(S) :: m \in X.$$

Property 4.46 illustrates that if a service can refuse the set X of events after performing σ, then a service can perform any other event that is not in X after performing σ.

Property 4.46 For each service S:

$$[\sigma, X] \in rp\text{-}failures(S) \;\wedge\; m \in (\Sigma) \setminus X \;\Rightarrow\; \sigma.m \in traces(S).$$

A *final* state q of service automaton S can be either a stable state or non-stable state. In case a final state q is stable, service S does not refuse a *final* event after having performing a trace σ and therefore S *can* terminate successfully with a final state after having performing σ. This is illustrated by Property 4.47.

Property 4.47 For each service S :

S can reach a final state after performing $\sigma \Leftrightarrow \exists[\sigma, X] \in \textit{rp-failures}(S) :: \textit{final} \notin X$.

By definition, a *responsive* service is a service that each stable τ-strongly connected component C enables a non-internal event from $\Sigma \cup \{\textit{final}\}$, that is, $act^*(Q_C) \neq \emptyset$. As the responsive refusal set $\textit{Refuse}(Q_C)$ at C is the complement of the set $act^*(Q_C)$, this is illustrated by Property 4.48.

Property 4.48 For each service S :

$$S \text{ is responsive } \Leftrightarrow \forall[\sigma, X] \in \textit{rp-failures}(S) :: X \neq \Sigma \cup \{\textit{final}\}.$$

We illustrate the set of all responsive failures of a service automaton with the following examples.

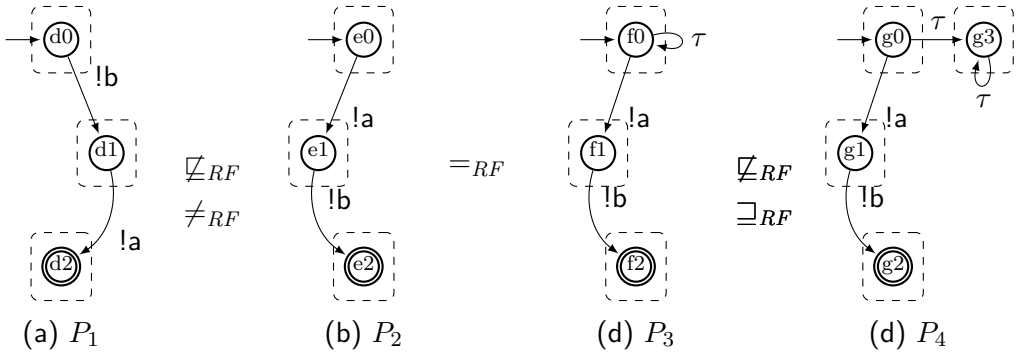


Figure 4.5.: Service P_1, P_2, P_3, and P_4 from Figure 4.4 illustrated with τ-strongly-connected components.

Example 4.49. Figure 4.5 illustrates services P_1, P_2, P_3, and P_4 from Figure 4.4 where a dashed rectangle denotes τ-strongly connect component of a service. The responsive failures of P_1, P_2, P_3, and P_4 are illustrated as follows.

$$\begin{aligned}
\textit{rp-failures}(P_1) &= \{\ [\epsilon, \{?a, \textit{final}\}], [?b, \{?b, \textit{final}\}], [?b?a, \{?a, ?b\}]\ \},\\
\textit{rp-failures}(P_2) &= \{\ [\epsilon, \{?b, \textit{final}\}], [?a, \{?a, \textit{final}\}], [?a?b, \{?a, ?b\}]\ \},\\
\textit{rp-failures}(P_3) &= \{\ [\epsilon, \{?b, \textit{final}\}], [?a, \{?a, \textit{final}\}], [?a?b, \{?a, ?b\}]\ \},\\
\textit{rp-failures}(P_4) &= \{\ [\epsilon, \{?a, ?b, \textit{final}\}], [?a, \{?a, \textit{final}\}], [?a?b, \{?a, ?b\}]\ \}.
\end{aligned}$$

We see that services P_2 and P_3 have the same set of responsive failures, whereas service P_4 contains more responsive failures than both P_2 and P_3. Note that the τ-strongly connect component that contains state $g3$ is stable but the τ-strongly connect component that contains state $g0$ is not. Therefore, the responsive failure $[\epsilon, \{?a, ?b, \mathit{final}\}]$ of service P_4 is calculated from the component containing state $g3$. $\lhd$

4.4.2. Responsive Failures Refinement and Equivalence

A service automaton R *refines* service automaton S *under responsive failures*, denoted by $S \sqsubseteq_{RF} R$ whenever every responsive failure of R is also a responsive failure of S.

Definition 4.50 (Responsive failures refinement).
A service automaton Q *refines* service automaton P *under responsive failures*, denoted by $S \sqsubseteq_{RF} R$, iff *rp-failures*$(S) \supseteq$ *rp-failures*(R). Then, the set of all services that refine service S under responsive failures is denoted by

$$\mathit{rf\text{-}refine}(S) \;=\; \{\; R \mid S \;\sqsubseteq_{RF} R \;\}.$$

The $\sqsubseteq_{RF}$ refinement relation is a *preorder* relation, as it is a reflexive and transitive relation.

Intuitively, $S \sqsubseteq_{RF} R$ means that every trace σ of R can be replayed by S and every responsive refusal set in R after having performed σ can also be refused in S after having performed the same trace σ. That is, R can neither accept nor refuse an event unless S does.

The responsive failures refinement $\sqsubseteq_{RF}$ induces an equivalence relation $=_{RF}$ that relates services with identical set of failures.

Definition 4.51 (Responsive failures equivalence).
Two service automata S and R are *equivalent under responsive failures*, denoted by $S \;=_{RF}\; R$, iff they have exactly the same set of responsive failures, i. e., *rp-failures*$(R) =$ *rp-failures*(S). Then, the set of all services that are equivalent to service S under responsive failures is denoted by

$$\mathit{rf\text{-}equiv}(S) \;=\; \{\; R \mid S \;=_{RF}\; R \;\}.$$

With the following lemma, we show that responsive failures preorder implies stable failures preorder.

Corollary 4.52 (Responsive failures preorder implies stable failures preorder).
For each message bound $k \in \mathbb{N}$ and each two k-responsively controllable service S and R with equivalent interface :

$$R \in \mathit{f\text{-}refine}(S) \quad \Rightarrow \quad R \in \mathit{rf\text{-}refine}(S).$$

Proof. Responsive failures extends stable failures by lifting up a refusal set at a stable state to a refusal set of a stable τ-strongly connected component. Consider state q in R and $[\sigma, X] \in \mathit{failures}(R)$ with $q_0 \stackrel{\sigma}{\Rightarrow}_R q$. In case q is a stable state, there is a stable τ-strongly connected component of R that contains only state q. In case q is a not a stable state, we have $X = \{\}$. For both cases, $[\sigma, X] \in \mathit{rp\text{-}failures}(R)$ also holds. Suppose $\mathit{failures}(R) \subseteq \mathit{failures}(S)$ holds, then $\mathit{rp\text{-}failures}(R) \subseteq \mathit{rp\text{-}failures}(S)$ follows. □

Nevertheless, the reverse of Corollary 4.52 does not hold.

With the following lemma, we show that a (complete) canonical responsive controller $\mathcal{C}_{rp,k}(S)$ and a compact canonical responsive controller $\widehat{\mathcal{C}}_{rp,k}(S)$, as introduced in Section 4.1.2, by construction are equivalent under responsive failures.

Corollary 4.53 (Canonical responsive controllers are equivalent under responsive failures).
For each message bound $k \in \mathbb{N}$ and each k-responsively controllable service S:

$$\mathcal{C}_{rp,k}(S) \;=_{RF}\; \widehat{\mathcal{C}}_{rp,k}(S).$$

Proof. The proof of this lemma follows from the proof of Lemma 4.31 for canonical deadlock-free controllers. This is because the canonical deadlock-free controller and the canonical responsive controller share common construction procedure (Definition 3.23 for the complete canonical controller and Definition 3.30 for the compact canonical controller). Though each construction procedure requires different inputs (i. e., either a deadlock-free operating guideline or a responsive operating guideline), the procedure canonically produces a canonical controller as an output from given operating guideline.

The responsive failures preorder is stronger than stable failures preorder (Corollary 4.52). Furthermore, the construction procedure of a canonical responsive controller never produces any stable τ-strongly connected component that contains more than one state, which is also a stable state in such a case. Thus, the proof of this lemma can be reconstructed straightforwardly from the proof of Lemma 4.31 for canonical deadlock-freedom controllers. □

Example 4.54. Consider service S_1 from Figure 4.1 and most permissive k-responsive controller $mp_{rp,k}(S_1)$ of service S_1 and the two canonical k-responsive controllers of service S_1; $\mathcal{C}_{rp,k}(S_1)$ and $\widehat{\mathcal{C}}_{rp,k}(S_1)$, from Figure 4.3. Their responsive failures are illustrated as follows.

$$\begin{aligned}
\mathit{rp\text{-}failures}(mp_{rp,k}(S_1)) = \{\ & [\epsilon, \{\mathit{final}\}], [!b, \{!b, \mathit{final}\}], [!b!a, \{!a, !b\}], \\
& [!a, \{!a, \mathit{final}\}], [!a!b, \{!a, !b\}]\ \}. \\
\mathit{rp\text{-}failures}(\mathcal{C}_{rp,k}(S_1)) = \{\ & [\epsilon, \{!a, \mathit{final}\}), [\epsilon, \{!b, \mathit{final}\}), [!b, \{!b, \mathit{final}\}], \\
& [!b!a, \{!a, !b\}], [!a, \{!a, \mathit{final}\}], [!a!b, \{!a, !b\}]\ \}. \\
\mathit{rp\text{-}failures}(\widehat{\mathcal{C}}_{rp,k}(S_1)) = {} & \mathit{rp\text{-}failures}(\mathcal{C}_{rp,k}(S_1)).
\end{aligned}$$

The three controllers have the same interface; $\mathcal{C}_{rp,k}(S_1)$ and $\widehat{\mathcal{C}}_{rp,k}(S_1)$ have the same set of responsive failures, this means, $\mathcal{C}_{rp,k}(S_1) =_{SF} \widehat{\mathcal{C}}_{rp,k}(S_1)$, whereas every responsive failure of $mp_{rp,k}(S_1)$ is included in a responsive failure of both $\mathcal{C}_{rp,k}(S_1)$ and $\widehat{\mathcal{C}}_{rp,k}(S_1)$, this means that $mp_{rp,k}(S_1)$ refines both $\mathcal{C}_{rp,k}(S_1)$ and $\widehat{\mathcal{C}}_{rp,k}(S_1)$ under stable failures.

In comparison to services P_1, P_2, P_3, and P_4 from Figure 4.4, we see from Example 4.49 that all responsive failures of each of the three services P_1, P_2, and P_3 are included in the set of all responsive failures of $\mathcal{C}_{rp,k}(S_1)$ (and of $\widehat{\mathcal{C}}_{rp,k}(S_1)$). Therefore, each of the three services P_1, P_2, and P_3 refines $\mathcal{C}_{rp,k}(S_1)$ (and $\widehat{\mathcal{C}}_{rp,k}(S_1)$) under responsive failures. However, this is not the case for service P_4, as the failure $(\epsilon, \{?a, ?b, \mathit{final}\})$ of P_4 is not described by any failure of $\mathcal{C}_{rp,k}(S_1)$ (and of $\widehat{\mathcal{C}}_{rp,k}(S_1)$). ◁

4.4.3. Relationship with Responsiveness

In this section, we illustrate a relationship between responsive failures and responsiveness. As the responsive failures semantics distinguishes between deadlock and successful termination, nevertheless, it does not distinguish between deadlock and divergence. We show that the responsive failures refinement relation between a canonical responsive controller and any other controller of a given service coincides with the responsive failures refinement.

The following example illustrates that there is no immediate coincidence between responsiveness inclusion and responsive failure refinement on asynchronously communicating services.

Example 4.55. Consider services P_1, P_2, P_3, and P_4 in Figure 4.4 for message bound $k = 1$ on each message channel. Service P_2 and service service P_3 are equivalent under responsive failures ($P_2 =_{RF} P_3$) whereas service P_3 refines service P_4 under responsive failures ($P_4 \sqsubseteq_{RF} P_3$). See also Example 4.54 . Nevertheless, services P_1, P_2 and P_3 are equivalent under k-responsiveness ($P_1 =_{rp,k} P_2 =_{rp,k} P_3$), whereas service P_4 is not k-responsively controllable.

Similarly for S_1, S_2, S_3, and S_4 in Figure 4.1 for $k = 1$. Service S_2 and service service S_3 are equivalent under responsive failures ($S_2 =_{RF} S_3$) whereas service S_3 refines service S_4 under responsive failures ($S_4 \sqsubseteq_{RF} S_3$). Nevertheless, services S_1, S_2, and S_3 are equivalent under k-responsiveness ($S_2 =_{rp,k} S_2 =_{rp,k} S_3$), whereas service S_4 is not k-responsively controllable. ◁

Similar to the relationship between stable failures refinement and deadlock freedom inclusion on the set of deadlock-free controllers, we observe that responsive failures refinement on a given service introduces more possibilities for a service to communicate with other interacting services, as the refined service refuses less events after having executed a trace.

Next, we prove that every pair of services that satisfies the responsive failures refinement also satisfies the substitution responsiveness inclusion.

Lemma 4.56 (Responsive failures refinement implies responsiveness inclusion).
For each message bound $k \in \mathbb{N}$ and each two k-responsively controllable services S and R with equivalent interface :

$$R \in \textit{rf-refine}(S) \;\Rightarrow\; R \;\in\; \textit{rp}_k\textit{Inclusion}(S).$$

Proof. Suppose $R \in \textit{rf-refine}(S)$, this means that $\textit{rp-failures}(R) \subseteq \textit{rp-failures}(S)$ by definition. Suppose $P \in \textit{rp}_k\textit{Controllers}(S)$. We will show that $P \in \textit{rp}_k\textit{Controllers}(R)$. Because controller is a symmetric notion (Proposition 2.27), we will show that $R \in \textit{rp}_k\textit{Controllers}(P)$.

Let $\textit{mp}_{\textit{rp},k}(P)^{\psi^*}$ be a k-responsive operating guideline of P.

Because controller is a symmetric notion (Proposition 2.27), $S \in \textit{rp}_k\textit{Controllers}(P)$ also holds. This means, S is structurally simulated by $\textit{mp}_{\textit{rp},k}(P)$ and, for an arbitrary simulated pair $[q_S, q_m]$ of state, state q_S structurally corresponds to $\psi^*(q_m)$ (Lemma 3.51). As structural simulation implies traces refinement, there exist (1) a trace σ_S in S with $q_{0S} \stackrel{\sigma_S}{\Rightarrow}_S q_S$ with state q_S in a stable τ-strongly connected component C_S of S such that $\sigma_S = \sigma_m$, and (2) a choice $ch \in \psi^*(q_m)$ with $ch = \textit{act}^*(\tau(q_S))$.

Let $[\sigma_R, X_R] \in \textit{rp-failures}(R)$. Because $\textit{rp-failures}(R) \subseteq \textit{rp-failures}(S)$ holds by assumption, we have $\textit{traces}(R) \subseteq \textit{traces}(S)$ and $[\sigma_R, X_R] \in \textit{rp-failures}(S)$ immediately follows. Because $\textit{traces}(R) \subseteq \textit{traces}(S)$, $\textit{traces}(R) \subseteq \textit{traces}(\textit{mp}_{\textit{rp},k}(P))$ follows. Because $\textit{mp}_{\textit{rp},k}(S)$ is deterministic, R is structurally simulated by $\textit{mp}_{\textit{rp},k}(P)$.

Let $[\sigma_S, X_S] \in \textit{rp-failures}(S)$ with $q_{0S} \stackrel{\sigma_S}{\Rightarrow} q_S$ such that $\sigma_R = \sigma_S$ and $X_S = \textit{Refuse}(q_{C_S})$.

As $[\sigma_R, X_R] \in \textit{rp-failures}(S)$, there exists a stable τ-strongly connected component C_R in R and a state $q_R \in Q_{C_R}$ in C_R such that $q_{0R} \stackrel{\sigma_R}{\Rightarrow} q_R$, $X_R = \textit{Refuse}(Q_{C_R})$, and $X_R \subseteq X_S$.

For each stable τ-strongly connected component C, we have $\textit{Refuse}(Q_C) = \Sigma \cup \{\textit{final}\} \setminus \textit{act}^*(Q_C)$. Because S and R are interface equivalent by assumption, it follows from $X_R \subseteq X_S$ that $\textit{act}^*(Q_{C_R}) \supseteq \textit{act}^*(Q_{C_S})$ must hold.

As $\sigma_R = \sigma_S$, the event e at q_R is enabled by C_R but not enabled by C_S (i. e., $e \in \textit{act}^*(Q_{C_R}) \setminus \textit{act}^*(Q_{C_S})$) and does not introduce an extra trace that is not described by S. As a τ-strongly connected component contains states in which each state is internally reachable from any other state within the same component (if there is any) by performing a sequence of internal τ events, there must be $ch' \in \psi^*(q_m)$ such that $ch' \supseteq ch$ and $ch' = \textit{act}^*(\tau(q_R))$.

This means, R is structurally simulated by $\textit{mp}_{\textit{rp},k}(P)$ and, for a simulated pair $[q_R, q_m]$ of states, state q_R structurally corresponds to $\psi^*(q_m)$. Thus, $S \in \textit{rp}_k\textit{Controllers}(P)$ holds (Lemma 3.69) and $P \in \textit{rp}_k\textit{Controllers}(S)$ follows (Proposition 2.27). That is, $\textit{rp}_k\textit{Controllers}(S) \subseteq \textit{rp}_k\textit{Controllers}(R)$ holds and $R \in \textit{rp}_k\textit{Inclusion}(S)$ follows by definition. □

Nevertheless, the reverse of Lemma 4.34 does not hold. Lemma 4.56 suggests that responsive failures refinement is finer than our responsiveness inclusion, as every service

that refines service S under responsive failures is a substitute for service S under k-responsiveness inclusion.

It is also important to notice that Lemma 4.56 holds only for k-responsively controllable services S and R.

As the responsive failures refinement and responsiveness inclusion are both preorder relations that induce an equivalence relation, we obtain the following corollary from Lemma 4.56.

Corollary 4.57 (Responsive failures equivalence implies responsiveness equivalence).
For each message bound $k \in \mathbb{N}$ and each two interface compatible services S and R :

$$R \in \textit{rf-equiv}(S) \;\Rightarrow\; R \;\in\; \textit{rp}_k\textit{Equiv}(S).$$

Proof. As the preorder induces the equivalence relation, the proof of this corollary follows from Lemma 4.56. □

Nevertheless, the reverse of Lemma 4.56 and of Corollary 4.57 do not hold (e. g., see Figure 4.4 and Figure 4.1 for counter examples). This means, the responsive failures refinement (and equivalence) is indeed finer than the responsiveness inclusion (and equivalence).

Example 4.58. Consider services S_2 and S_3 in Figure 4.1. Service S_2 and S_3 are equivalent under responsive failures ($S_2 =_{RF} S_3$). As suggested by Corollary 4.57, services S_2 and S_3 are also equivalent under k-responsiveness ($S_2 =_{rp,k} S_3$) for message bound $k = 1$ on each message channel.

Similar for services P_2 and P_3 in Figure 4.4. Services P_2 and P_3 are equivalent under responsive failures ($P_2 =_{RF} P_3$). Corollary 4.57 suggests that services P_2 and P_3 are also equivalent under k-responsiveness ($P_2 =_{rp,k} P_3$) for $k = 1$.

Given two services that are equivalent under k-responsiveness, for instance, services P_1 and P_2 with $P_1 =_{rp,k} P_2$ for $k = 1$, it is not the case that P_1 and P_2 are equivalent under k-responsiveness, however. ◁

Though the responsive failures refinement and the responsiveness inclusion do not coincide, the set of controllers of a given service S that is equipped with responsiveness preorder and with the responsive failures refinement has one maximal element in common, that is, a canonical k-responsive controller $\mathcal{C}_{rp,k}(S)$ of service S.

With Theorem 4.59, we show that there is a coincidence between responsiveness inclusion and the responsive failures refinement via one distinguished maximal element of the set of responsive controllers of service S, that is, a canonical k-responsive controller $\mathcal{C}_{df,k}(S)$ of S.

Theorem 4.59 (Relationship between responsive failures refined services and responsive controllers).
For each message bound $k \in \mathbb{N}$ and each k-responsively controllable services S :

$$\textit{rf-refine}(\mathcal{C}_{rp,k}(S)) = rp_k\textit{Controllers}(S).$$

Proof. By construction, a canonical k-responsive controller $\mathcal{C}_{rp,k}(S)$ of S is derived from the k-responsive operating guideline $mp_{rp,k}(S)^{\psi^*}$ of S and they are interface equivalent. We prove this lemma in two directions.

$\Rightarrow$: Suppose $P \in \textit{rf-refine}(\mathcal{C}_{rp,k}(S))$, that is, $\textit{rp-failures}(P) \subseteq \textit{rp-failures}(\mathcal{C}_{df,k}(S))$ holds. Let $\mathcal{C}_{rp,k}(S)$ be constructed from the operating guideline $mp_{rp,k}(S)^{\psi^*}$ according to Definition 4.6. We will show that P structurally complies with $mp_{rp,k}(S)^{\psi^*}$.

As $\textit{rp-failures}(P) \subseteq \textit{rp-failures}(\mathcal{C}_{rp,k}(S))$, every trace in P can be replayed by $\mathcal{C}_{rp,k}(S)$. As $\textit{traces}(mp_{rp,k}(S)) = \textit{traces}(\mathcal{C}_{rp,k}(S))$ holds (Property 4.18), every trace in P can also be replayed by $mp_{rp,k}(S)$. As $mp_{rp,k}(S)$ is a deterministic service automaton, it follows that there is a structural simulation relation between P and $mp_{rp,k}(S)$ such that P is structurally simulated by $mp_{rp,k}(S)$.

Let $[\sigma_P, X_P] \in \textit{rp-failures}(P) \subseteq \textit{rp-failures}(\mathcal{C}_{rp,k}(S))$. Let C_P be a τ-strongly connected component C_P in P with state q_P contained in C_P and $q_{0P} \stackrel{\sigma_P}{\Rightarrow}_P q_P$. Then, there exists a τ-strongly connected component C in $\mathcal{C}_{rp,k}(S)$ with state q_m contained in C and $q_{0C} \stackrel{\sigma_P}{\Rightarrow}_C q_C$.

We consider a simulated pair $[q_P, q_m]$ of a state q_P in P that is structurally simulated by a state q_m in $mp_{rp,k}(S)$.

- In case C_P is a stable τ-strongly connected component in P, we have $X_P = \textit{Refuse}(Q_{C_P})$. As the refusal set is the complement set of the set of all enabled events by definition, it follows that $Y_P = \textit{act}^*(q_P) = \Sigma_P \cup \{\textit{final}\} \setminus \textit{Refuse}(Q_{C_P})$. Because $[\sigma_P, X_P] \in \textit{rp-failures}(\mathcal{C}_{rp,k}(S))$ holds, then there exists a stable τ-strongly connected component C in $\mathcal{C}_{rp,k}(S)$ such that (1) every state q_C in C is structurally simulated by state q_m, and (2) C refuses the same set X_P of events as by component C_P and therefore C offers the same enable events Y_P as they are offered by C_P.
  Because $\mathcal{C}_{rp,k}(S)$ contains a nondeterministic internal alternatives of all choices from $\psi^*(q_m)$ by construction, it follows that every state in C structurally corresponds to $\psi^*(q_m)$. Thus, also every state in C_P also structurally corresponds to $\psi^*(q_m)$.
- In case the component C_P is not stable in P, this means there exists state $q'_P \in \tau(q_P)$ that is reachable from q_P via τ transition in P such that q'_P is not in C_P. For each $q'_P \in \tau(q_P)$ we have $[q'_P, q_m]$ as a structurally simulated pair of states. Consider $q'_P \in \tau(q_P)$ that is contained in a stable τ-strongly connected component C'_P. It follows that C refuses the same set of events as component C'_P and therefore C offers the same enable events as they are offered by C'_P.

This implies that for every state q'' in C'_P there exists a choice $ch \in \psi^*(q_m)$ such that we have $ch = act^*(\tau(q'')) = act^*(\tau(q))$ holds for each q in the component C. This means q_P also structurally corresponds to $\psi^*(q_m)$.

Therefore, P structurally complies with $mp_{rp,k}(S)^{\psi^*}$ and $P \in rp_k Controllers(S))$ holds (Lemma 3.69).

$\Leftarrow$: Suppose $P \in rp_k Controllers(S)$ holds, that is, P structurally complies with the operating guideline $mp_{rp,k}(S)^{\psi^*}$ of S (Lemma 3.69).

Consider $[\sigma, X] \in$ *rp-failures*(P). We will show that $[\sigma, X] \in$ *rp-failures*$(\mathcal{C}_{df,k}(S))$ holds.

As $\mathcal{C}_{rp,k}(S)$ is a k-responsive controller of S (Corollary 4.7) and $mp_{rp,k}(S)$ is a most permissive k-responsive controller of S, $mp_{rp,k}(S)$ structurally simulates both $\mathcal{C}_{rp,k}(S)$ and P (Lemma 3.66). By construction of $\mathcal{C}_{rp,k}(S)$, *traces*$(mp_{rp,k}(S)) =$ *traces*$(\mathcal{C}_{rp,k}(S))$ holds (Property 4.18). This means, $\sigma \in$ *traces*$(mp_{rp,k}(S))$ whenever $\sigma \in$ *traces*$(\mathcal{C}_{rp,k}(S))$. Then, σ is also a trace of $\mathcal{C}_{rp,k}(S)$.

Consider a state q_γ in $\mathcal{C}_{rp,k}(S)$ with $q_0 \stackrel{\sigma}{\Rightarrow} q_\gamma$.

Consider a pair of structurally simulated state $[q_P, q_m]$ of a state q_P in P by a state q_m in $\mathcal{C}_{rp,k}(S)$. Because $P \in rp_k Controllers(S)$ by assumption, q_P structurally corresponds to $\psi^*(q_m)$ (Lemma 3.69).

- Consider $ch = \psi^*_k(q_m)$ such that there exists state q_P in P with $ch = act^*(\tau(q_P))$. In case q_P is contained in a stable τ-strongly connected component, it follows that $ch = (\Sigma \cup \{final\})\backslash X$. Otherwise, there exists $q_P \stackrel{\tau}{\Rightarrow}_P q'_P$ such that q'_P is contained in a stable τ-strongly connected component and $ch = act^*(\tau(q'_P))$ holds.
  As $ch = \psi^*_k(q_m)$, we consider state q_γ in $\mathcal{C}_{rp,k}(S)$ with $ch = enable(q_\gamma)$ by construction of $\mathcal{C}_{rp,k}(S)$. Because $mp_{rp,k}(S)$ is a nondeterministic τ-free service automaton, $\tau \notin ch$ and $X = (\Sigma \cup \{final\})\backslash ch$ follows.
- Consider $ch = \psi^*_k(q_m)$ such that $ch = \emptyset$. This means that $\mathcal{K}(q_m) = \emptyset$ and q_m is never covered in the composition with S. In case that q_P is a divergent state and C is a stable τ-strongly connected component that contains q_P, state q_P structurally corresponds to ch and $act^*(\tau(Q_C)) = \emptyset$. It follows that $X = \Sigma \cup \{final\} \setminus act^*(\tau(Q_C)) = \Sigma \cup \{final\}$. In such case there exists a divergent q_γ by construction.

Either case, it means that $[\sigma, X]$ is also a failure of $\mathcal{C}_{rp,k}(S)$ at state q_γ. Therefore, $[\sigma, X] \in$ *rp-failures*$(\mathcal{C}_{rp,k}(S))$ and *rp-failures*$(\mathcal{C}_{rp,k}(S)) \supseteq$ *rp-failures*(P) holds. Thus, $\mathcal{C}_{rp,k}(P) \sqsubseteq_{RF} P$ follows by definition.

Thus, this theorem holds. □

Theorem 4.59 suggests that the set of all responsive controllers of service S coincides with the set of all services that refine a canonical k-responsive controller $\mathcal{C}_{df,k}(S)$ of S under responsive failures.

Similarly to a canonical k-deadlock-free controller, a canonical k-responsive controller $\mathcal{C}_{df,k}(S)$ of S, as stated in Theorem 4.59, can be replaced only by an arbitrary k-responsive controller of S that is equivalent to a canonical k-responsive controller $\mathcal{C}_{df,k}(S)$ of S under responsive failures, such as a compact canonical k-responsive controller of S (cf. Corollary 4.53). This is due to the distinguished property of a canonical k-responsive controller $\mathcal{C}_{df,k}(S)$ of S.

We illustrate the value of Theorem 4.59 with the following example.

Example 4.60. Consider services P_1, P_2, P_3, and P_4 in Figure 4.4, the canonical k-responsive controller $\mathcal{C}_{rp,k}(S_1)$ of S_1 in Figure 4.3, and message bound $k = 1$ on each message channel.

We see that each service P_1, P_2 and P_3 refines the canonical k-responsive controller $\mathcal{C}_{rp,k}(S_1)$ of S_1 under responsive failures. However, service P_4 does not. This is because service P_4 refuses $\{?a, ?b, \mathit{final}\}$ at the initial empty trace ϵ, and $[\epsilon, \{?a, ?b, \mathit{final}\}]$ is not a responsive failure of $\mathcal{C}_{rp,k}(S_1)$.

This means, services P_1, P_2, and P_3 are k-responsive controllers of service S_1 illustrated in Figure 4.1 (also k-responsive controllers of service S_2 as S_1 and S_2 are equivalent under k-responsiveness). Though service P_4 is neither a k-responsive controller of S_1 nor a k-responsive controller of S_3. ◁

Next, we prove in Lemma 4.61 that we can decide if service T is a substitute for service S under responsiveness preorder by checking if a canonical k-responsive controller $\mathcal{C}_{rp,k}(S)$ of S refines a canonical k-responsive controller $\mathcal{C}_{rp,k}(T)$ of T under responsive failures.

Lemma 4.61 (Responsive failures refinement of canonical controllers).
For each message bound $k \in \mathbb{N}$ and each two interface equivalent services S and T that are k-responsively controllable :

$$T \sqsubseteq_{rp,k} S \Leftrightarrow \mathcal{C}_{rp,k}(T) \sqsubseteq_{RF} \mathcal{C}_{rp,k}(S).$$

Proof. Given :

$$\begin{aligned} T \sqsubseteq_{rp,k} S &\Leftrightarrow \{\text{by Definition 2.38}\} \\ &\quad rp_k\mathit{Controllers}(T) \supseteq rp_k\mathit{Controllers}(T) \\ &\Leftrightarrow \{\text{by applying Thoerem 4.59 twice}\} \\ &\quad \mathit{rf\text{-}refine}(\mathcal{C}_{rp,k}(T)) \supseteq \mathit{rf\text{-}refine}(\mathcal{C}_{rp,k}(S)) \\ &\Leftrightarrow \{\text{by Definition 4.50 and set implication}\} \\ &\quad \mathcal{C}_{rp,k}(T) \sqsubseteq_{SF} \mathcal{C}_{rp,k}(S). \end{aligned}$$

Consequently, the lemma holds. □

In Corollary 4.62, we prove that we can decide if services S and T are responsively equivalent by checking if two canonical k-responsive controllers, $\mathcal{C}_{rp,k}(S)$ of S and $\mathcal{C}_{rp,k}(T)$ of T, are equivalent under responsive failures.

Corollary 4.62 (Responsive failures equivalence of canonical controllers). For each message bound $k \in \mathbb{N}$ and each two interface equivalent services S and T that are k-responsively controllable :

$$T =_{rp,k} S \;\Leftrightarrow\; \mathcal{C}_{rp,k}(T) \;=_{RF}\; \mathcal{C}_{rp,k}(S).$$

Proof. Follows from Lemma 4.61. □

With the aforementioned properties, a canonical k-responsive controller $\mathcal{C}_{rp,k}(S)$ of a given service S can be regarded as a *behavioral specification* of all k-responsive controllers of service S for $k \in \mathbb{N}$. For checking whether an implementation (e. g., service P) *conforms* to the specification, we apply the responsive failures refinement relation as the conformance relation. For a service P that refines the specification $\mathcal{C}_{rp,k}(S)$ under responsive failures, we conclude that service P is indeed an implementation (of the specification $\mathcal{C}_{rp,k}(S)$) of S, meaning P can interact responsively with S. We discuss the respective applications in Chapter 7.

4.5. Controller Synthesis by Transformation

In the previous sections, we have illustrated a coincidence between the stable failures refinement (and equivalence) and deadlock freedom preorder (and equivalence) that can only established via a canonical deadlock-free controller of a given service. Though the stable failures refinement (and equivalence) is finer than deadlock freedom preorder (and equivalence), we can employ the order posed by stable failures semantics on the set of deadlock-free controllers of a given service in a way that a canonical deadlock-free controller of a given service is larger (or equal) in the stable failures preorder than (or to) every other controller of a given service. Similarly for the responsive failures refinement (and equivalence) and responsiveness preorder (and equivalence) established via a canonical responsive controller of a given service.

In this section, we propose a set of transformation rules each of which preserves either refinement or equivalence of both stable failures and responsive failures. These rules are defined on service automata in the style of Murata rules [Murata, 1989]. As the stable failures refinement is finer than deadlock freedom inclusion (cf. Section 4.3.3), the rule that preserves either stable failures refinement or equivalence, by implication, it also preserves deadlock freedom inclusion or equivalence respectively. Similarly to the rules that preserve responsive failures refinement, as responsive failures refinement is finer than responsiveness inclusion (cf. Section 4.4.3).

The set of transformation rules presented in the section is limited and not complete; there is no guarantee that all possible services will be covered by means of transformation. Nevertheless, the application of transformation rules is highly useful in practice. In order to synthesize a controller of a given service, one possible method is to first synthesize a canonical controller from a given service, then to apply a sequence of transformation rules, as introduced in this section, to a canonical controller in order to synthesize a

controller of a given service that suits to a specific requirement (for instance, to synthesize a controller with a least number of states). For this purpose, we first define, for each state q of a service automaton, the set of all events that are immediately enabled before reaching state q as follows.

Definition 4.63 (Incoming events).
For a state q of a service automaton S, we denote the set of all *enabled events* that are immediately enabled before reaching state q by $\mathit{incoming}(q) = \{e \mid q' \xrightarrow{e} q\} \cup \{\mathit{final} \mid q' \in \Omega \ \wedge \ q' \xrightarrow{e} q\}$.

In the remainder of this section, we present ten transformation rules. Section 4.5.1 presents the first rule $SF(1)$ which preserves stable failures refinement. Section 4.5.2 presents the rule $RF(1)$ which preserves refinement of responsive failures refinement defined as a variant of rule $SF(1)$. Section 4.5.3 illustrates six transformation rules; $F(1), F(2), F(3), F(4), F(5)$, and $F(6)$, each preserves refinement and/or equivalence under stable failures and responsive failures. Section 4.5.4 shows two transformation rules; $R(1)$ and $R(2)$, each preserves both deadlock freedom and responsiveness under inclusion and equivalence.

4.5.1. Stable Failures Transformation Rules

The stable-failures-preserving rule $SF(1)$ is defined as follows.

Definition 4.64 (Rule $SF(1)$).
Rule $SF(1)$ as shown in Figure 4.6 aims at removing other paths that are alternatives to an invisible τ-transition.

Assumption: Let $n \in \mathbb{N}$ where $n \geq 1$. Let S and R be two interface equivalent service automata such that :

- S contains two states s_1 and s_2 with a transition $s_1 \xrightarrow{\tau} s_2$ and $\{e_1, \ldots, e_n, \tau\} \subseteq \mathit{enable}(s_1)$,
- R contains two states r_1 and s_2 with a transition $r_1 \xrightarrow{\tau} s_2$.

Application: Transform S into R by replacing state s_1 with state r_1 and by doing so remove all transitions labeled with events in $\{e_1, \ldots, e_n\}$ and their subsequent paths that are not joined with any other path from s_1 such that

- each state that is reachable from s_1 via other paths than the removed paths starting with events in $\{e_1, \ldots, e_n\}$ must be reachable from state r_1, and
- $\mathit{enable}(r_1) \cup \{e_1, \ldots, e_n\} = \mathit{enable}(s_1)$ must hold.

The following lemma justifies rule $SF(1)$.

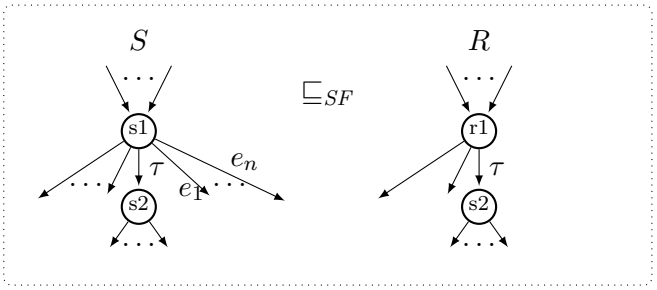


Figure 4.6.: Transformation rule $SF(1)$ that preserves refinement of stable failures.

Lemma 4.65 (Justification of rule $SF(1)$).
For each two interface equivalent services S and R that are related by rule $SF(1)$:

$$S \sqsubseteq_{SF} R.$$

Proof. Consider $[\sigma, X] \in \mathit{failures}(R)$. By applying rule $SF(1)$ on S, we transform S into R by removing all events in $\{e_1, \ldots, e_n\}$ and their subsequent paths that are not joined with any other path from from s_1 in S. This means, S can replay every trace in R and S can refuse every refusal set that R refuses. Thus, $[\sigma, X] \in \mathit{failures}(S)$ holds. □

4.5.2. Responsive Failures Transformation Rules

The following rule $RF(1)$ is a variant of Rule $SF(1)$ defined by Definition 4.64 by restricting one assumption for responsive failures property.

Definition 4.66 (Rule $RF(1)$).
Rule $RF(1)$ as shown in Figure 4.7 aims at removing other paths that are alternatives to an invisible τ-transition.

Assumption: Let $n \in \mathbb{N}$ and $n \geq 1$. Let S and R be two interface equivalent service automata such that :

- R contains two states r_1 and s_2 with a transition $r_1 \xrightarrow{\tau} s_2$ and $\tau \notin \mathit{enable}(s_2)$,
- S contains two states s_1 and s_2 with a transition $s_1 \xrightarrow{\tau} s_2$ and $\{e_1, \ldots, e_n, \tau\} \subseteq \mathit{enable}(s_1)$ and $\tau \notin \mathit{enable}(s_2)$.

Application: Transform S into R by replacing state s_1 by r_1 and by doing so remove all transition labeled with events in $\{e_1, \ldots, e_n\}$ and their subsequent paths that are not joined with any other path from state s_1 such that

- each state that is reachable from s_1 via other paths than the removed paths starting with events in $\{e_1, \ldots, e_n\}$ must be reachable from r_1, and
- $\mathit{enable}(r_1) \cup \{e_1, \ldots, e_n\} = \mathit{enable}(s_1)$ must hold.

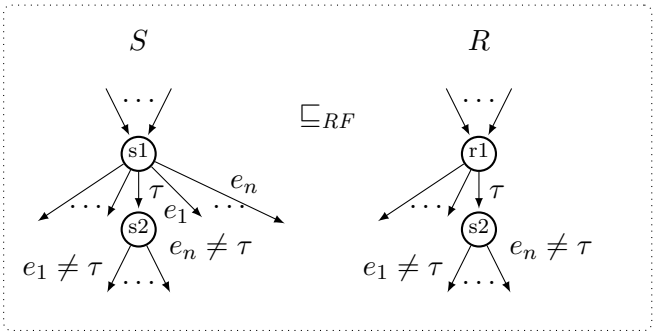


Figure 4.7.: Responsive failures transformation rule $RF(1)$.

The following lemma justifies rule $RF(1)$.

Lemma 4.67 (Justification of rule $RF(1)$).
For each two interface equivalent services S and R that are related by Rule $RF(1)$:

$$S \sqsubseteq_{RF} R.$$

Proof. Let $[\sigma, X] \in$ *rp-failures*(R). By applying rule $RF(1)$ on R, we transform S into R by removing all events in $\{e_1, \ldots, e_n\}$ and their subsequent paths that are not joined with any other path from q_1 in R. Therefore, R can replay every trace in P. As $\tau \notin$ *enable*(s_2), state s_2 is not a divergent state by definition. Therefore, after having executed each trace R it will never refuse any event that S does not refuse. Thus, $[\sigma, X] \in$ *rp-failures*(S) holds. □

4.5.3. Common Transformation Rules

In this section, we presents six transformation rules; $F(1)$, $F(2)$, $F(3)$, $F(4)$, $F(5)$, and $F(6)$, each can be used to transform a given service into anther service that refines a given service under both stable failures and responsive failures and/or equivalence.

First, we define rule $F(1)$ as follows.

Definition 4.68 (Rule $F(1)$).
Rule $F(1)$ as shown in Figure 4.8 aims at either removing or inserting an intermediate invisible τ-transition.

Assumption: Let S and R be two interface equivalence service automata such that :

- S contains two states s_1 and s_2 with a transition $s_1 \xrightarrow{\tau} s_2$,
- R contains state r_1.

Application(s): There are two possible applications for Rule $F(1)$;
either

1. transform S into R by replacing intermediate transition $s_1 \xrightarrow{\tau} s_2$ with r_1; or
2. transform R into S by replacing r_1 with intermediate transition $S_1 \xrightarrow{\tau} S_2$;

such that $\mathit{incoming}(s_1) = \mathit{incoming}(r_1)$ and $\mathit{enable}(s_1) = \mathit{enable}(r_2)$ must hold.

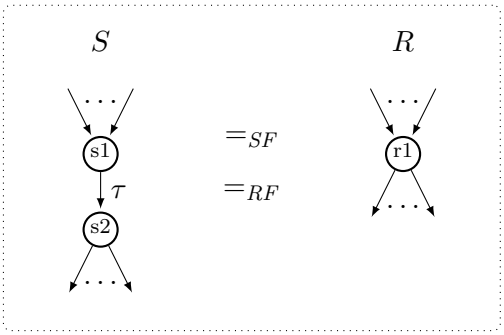


Figure 4.8.: Transformation rule $F(1)$ that preserves equivalence of both stable failures and responsive failures.

The following lemma justifies rule $F(1)$.

Lemma 4.69 (Justification of rule $F(1)$).
For each two interface equivalent services S and R that are related by Rule $F(1)$:

$$S =_{SF} R \text{ and } S =_{RF} R.$$

Proof. By applying rule $F(1)$ on S, we transform S into R by replacing the intermediate transition $s_1 \xrightarrow{\tau} s_2$ with state t_1. By applying rule $F(1)$ on R, we transform R into S by inserting one τ loop. For both cases, every incoming event and outgoing event are preserved under transformation. The intermediate τ transition and the τ loop neither introduce nor nullify any trace and any refusal set of events before the application. Thus, we conclude that $S =_{SF} R$ and $S =_{RF} R$ holds. □

Example 4.70. Figure 4.9 illustrates different order of applying Rule 4.64 and Rule 4.68 on service M (shown in the right-most sub-figure). Rule $SF(1)$ transforms service M into either service N_1 or service O_1, depending on which branches of M is being thrown away during the application. Service N_1 can be transformed further into service N_2 by applying rule $F(1)$ to eliminate a τ-transition; similarly, service O_1 can be transformed into service O_2 by applying $F(1)$. Each transformation step guarantees that the transformed service refines the service before transformation under stable failures.

In contrast to an application of rule $SF(1)$, we cannot apply rule $RF(1)$ to transform M into O_1 due to an assumption of $RF(1)$. ◁

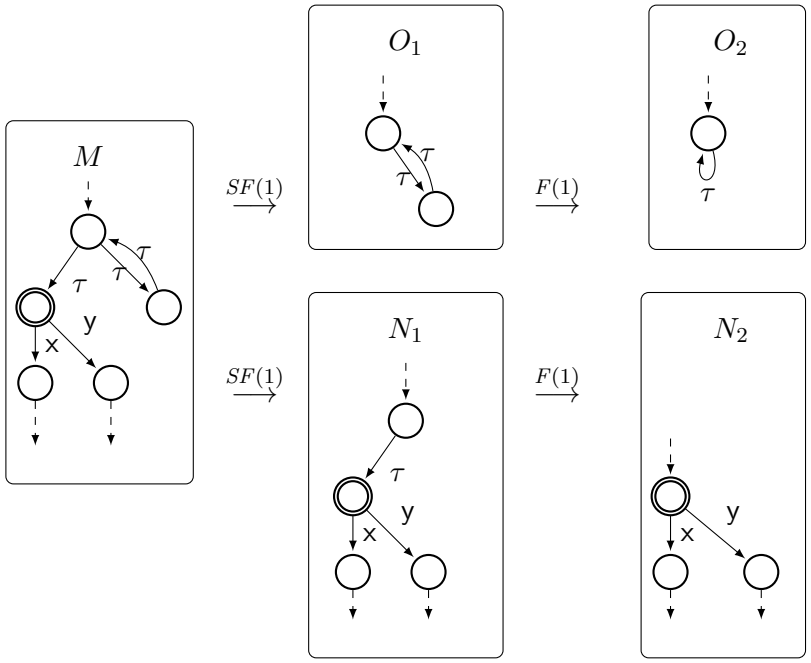


Figure 4.9.: Fragments of two services N_2 and O_2, each can be transformed from service M by applying a number of transformation rules.

Next, we define rule $F(2)$ as follows.

Definition 4.71 (Rule $F(2)$).
Rule $F(2)$ as shown in Figure 4.10 aims at either extending a τ-loop and some enabled events with an intermediate τ transition or removing an intermediate τ-transition.

Assumption: Let $n \in \mathbb{N}$ where $0 \leq m$, $1 < n$, and $m < n$. Let S and R be two interface equivalent service automata such that :

- S contains state s_1 with the transition $s_1 \xrightarrow{\tau} s_1$ such that $e_1, \ldots, e_n \in \mathit{enable}(s_1)$,
- R contains two states r_1 and r_2 with two transitions $r_1 \xrightarrow{\tau} r_2$, and $r_2 \xrightarrow{\tau} r_1$ such that $\mathit{enable}(r_1) = \{e_1, \ldots, e_m, \tau\}$ and $\mathit{enable}(r_2) = \{e_{m+1}, \ldots, e_n, \tau\}$.

Application(s): There are two possible applications for Rule $F(2)$; either

1. transform S into R by replacing the transition $s_1 \xrightarrow{\tau} s_1$ by the two transitions, $r_1 \xrightarrow{\tau} r_2$, and $r_2 \xrightarrow{\tau} r_1$;
2. transform S into R by replacing the two transitions, $s_1 \xrightarrow{\tau} s_2$ and $s_2 \xrightarrow{\tau} s_1$, by the transition $r_1 \xrightarrow{\tau} r_1$; or

such that $\mathit{incoming}(s_1) = \mathit{incoming}(r_1)$ and $\mathit{enable}(s_1) = \mathit{enable}(r_1) \cup \mathit{enable}(r_2)$ must hold for each application.

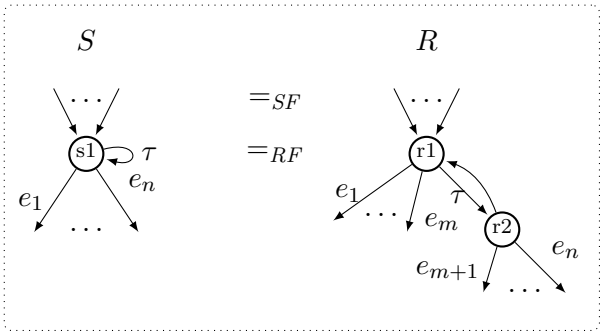


Figure 4.10.: Transformation rule $F(2)$ that preserves equivalence of both stable failures and responsive failures.

The following lemma justifies rule $F(2)$.

Lemma 4.72 (Justification of rule $F(2)$).
For each two interface equivalent services S and R that are related by Rule $F(2)$:

$$S =_{SF} R \text{ and } S =_{RF} R.$$

Proof. By applying rule $F(2)$ on S, splitting the set of outgoing transitions at state s_1 labeled over two state r_1 and r_2 neither introduces nor nullifies any trace before the application. By applying rule $F(2)$ on R, merging two sets of outgoing transitions at state r_1 and r_2 neither introduces nor nullifies any trace before the application.

$S =_{SF} R$: state s_1 is not a stable state as well as neither state r_1 nor state r_2 is a stable state. This means, the application of the rule in both directions neither introduces nor nullifies any refusal set of events before the application.

$S =_{RF} R$: Suppose state s_1 is in a stable τ-strongly component C_S in S. Suppose states r_1 and r_2 are in a stable τ-strongly component C_R in R. Clearly, C_S and C_R have the same refusal set.

Thus, $S =_{SF} R$ and $S =_{RF} R$ holds. □

Next, we define rule $F(3)$ as follows.

Definition 4.73 (Rule $F(3)$).
Rule $F(3)$ as shown in Figure 4.11 aims at either removing or inserting a τ-loop transition.

Assumption: Let S and R be two interface equivalent service automata such that :

– S contains a state S_1,

– R contains state r_1 and a transition $r_1 \xrightarrow{\tau} r_1$.

Application(s): There are two possible applications for Rule $F(3)$; either

1. transform S into R by replacing state s_1 by state r_1 and transition $r_1 \xrightarrow{\tau} r_1$; or
2. transform R into S by replacing state r_1 and the transition $r_1 \xrightarrow{\tau} r_1$ by state s_1;

such that $\mathit{incoming}(r_1) = \mathit{incoming}(s_1)$ and $\mathit{enable}(r_1) \cup \{\tau\} = \mathit{enable}(s_1)$ must hold for each application.

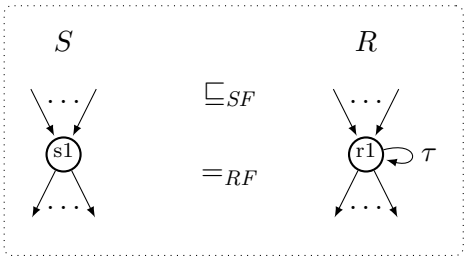


Figure 4.11.: Transformation rule $F(3)$ that preserves equivalence of stable failures and refinement of both stable failures and responsive failures.

Rule $F(3)$ is defined in the similar sense as e. g., a rule for adding a τ loop in Murata [1989], van der Aalst and Basten [2002]. Interestingly, the rule $F(3)$ preserves stable failures refinement in both directions, but preserves responsive failures refinement in only one direction. The following lemma justifies rule $F(3)$.

Lemma 4.74 (Justification of rule $F(3)$).
For each two interface equivalent services S and R that are related by Rule $F(3)$:

$$S \sqsubseteq_{SF} R \text{ and } S =_{RF} R.$$

Proof. By applying rule $F(3)$ on S, inserting a τ transition neither introduces nor nullifies any trace before the application. By applying rule $F(4)$ on R, removing a τ transition also neither introduces nor nullifies any trace before the application.

$S \sqsubseteq_{SF} R$: Clearly, state r_1 is not a stable state, whereas this is not necessarily the case for state s_1. This means, R does not refuse any event at state r_1, but it is possible that state s_1 is a stable state, and therefore, refuses the set of events that are not enabled at s_1. That is, $S \sqsubseteq_{SF} R$ holds.

$S =_{RF} R$: Suppose state s_1 is in a stable τ-strongly connected component C_S in S. Suppose state r_1 is in a stable τ-strongly connected component C_R in R. Clearly, C_S and C_R have the same refusal set. That is, $S =_{RF} R$ holds.

Thus, $S \sqsubseteq_{SF} R$ and $S =_{RF} R$ holds. □

Nevertheless, applying rule $F(3)$ to transform R into S does not guarantee that S can refine R under stable failures. This is because the removing τ transition from R possibly introduces a refusal set at state s_1 that is never refuses at state r_1.

Next, we define rule $F(4)$ as follows.

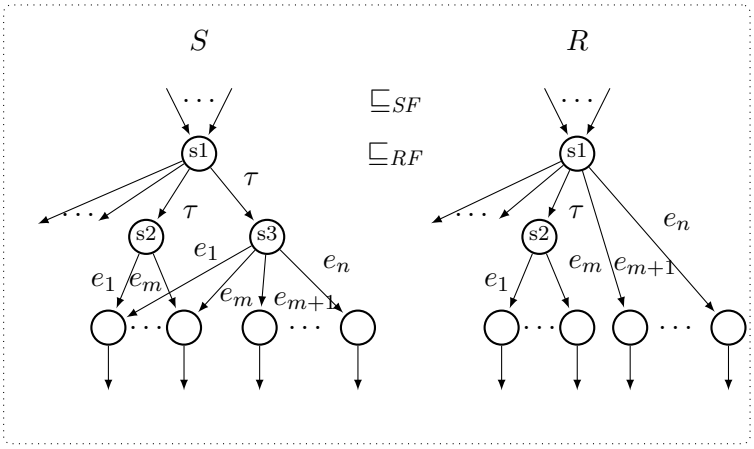


Figure 4.12.: Transformation rule $F(4)$ that preserves equivalence of stable failures.

Definition 4.75 (Rule $F(4)$).
Rule $F(4)$ as shown in Figure 4.12 aims at reducing one outgoing τ transition at a state q which offers redundant events at another state reachable from q.

Assumption: Let $m, n \in \mathbb{N}$ and $m, n \geq 1$. Let S and R be two interface equivalent service automata such that :

- S contains three states s_1, s_2, and s_3 with two transitions $s_1 \xrightarrow{\tau} s_2$ and $s_1 \xrightarrow{\tau} s_3$ such that $\mathit{enable}(s_2) \neq \{\tau\}$ and $\mathit{enable}(s_3) = \{e_1, \ldots, e_n\}$,
- R contains two states r_1 and s_2 with a transition $r_1 \xrightarrow{\tau} s_2$ such that $e_1, \ldots, e_n \in \mathit{enable}(r_1)$.

Application: Transform S into R by replacing the τ transition $s_1 \xrightarrow{\tau} s_2$ by state r_1 and by doing so remove all transitions labeled with events $\{e_{m+1}, \ldots, e_n\}$ and their subsequent paths that are not joined with any other path from states s_1 such that $\mathit{incoming}(s_1) = \mathit{incoming}(r_1)$ and $\mathit{enable}(r_1) = \mathit{enable}(s_1) \cup \{e_1, \ldots, e_n, \tau\}$ must hold.

The following lemma justifies rule $F(4)$.

Lemma 4.76 (Justification of rule $F(4)$).
For each two interface equivalent services S and R that are related by Rule $F(4)$:

$$S \sqsubseteq_{SF} R \text{ and } S \sqsubseteq_{RF} R.$$

Proof. By applying rule $F(4)$ on S, every trace of R is clearly also a trace of S.

$S \sqsubseteq_{SF} R$: State s_3 is not a stable state and neither is state r_1. Let σ be a trace of R at state r_1. It follows that for every event that R can refuse after having executed σ, S can also refuse such a event after having executed the same trace.

$S \sqsubseteq_{RF} R$: Suppose states s_1 and r_1 are in stable τ-strongly components C_S in S and C_R in R respectively. Clearly, for every event that R refuses at C_R, S can refuse the same event at C_S after having executed the same trace.

Thus, we conclude that $S \sqsubseteq_{SF} R$ and $S \sqsubseteq_{RF} R$ holds. □

Nevertheless, applying rule $F(4)$ to transform R into S does not guarantee that S can refine R under both stable failures and responsive failures. This is because the introduced paths in S possibly introduce new traces into S.

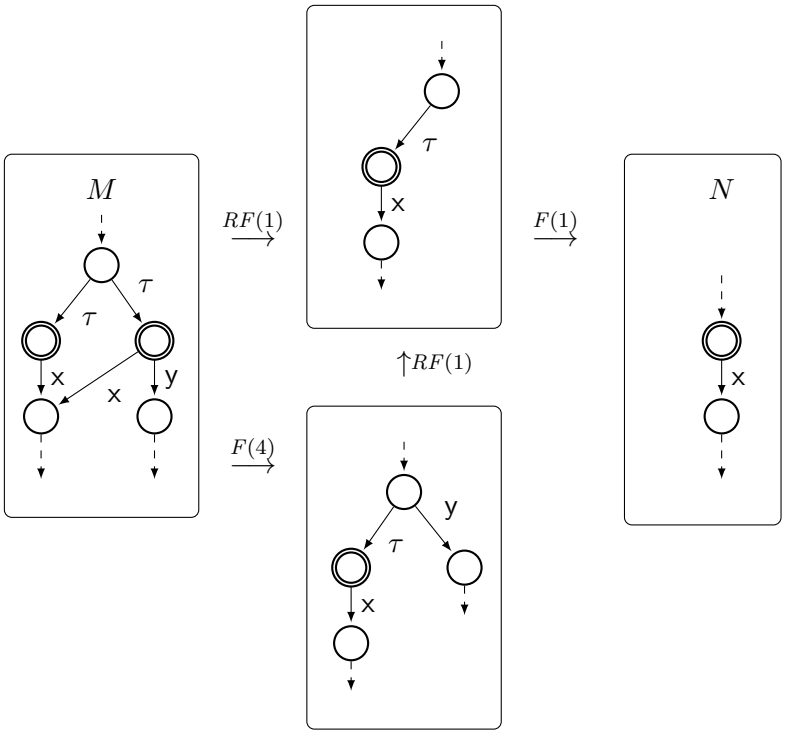


Figure 4.13.: Fragment of service N in which service N can be transformed from service M by applying a number of transformation rules.

Example 4.77. Figure 4.13 shows different orders of applying various transformation rules, each rule preserves refinement of stable failures. Service M on the left-most can be transformed eventually into service N on the right-most by applying a number of deadlock-free-preserving transformation rules. The application of these rules guarantees that N refines M under stable failures. ◁

Next, we define rule $F(5)$ as follows.

Definition 4.78 (Rule $F(5)$).
Rule $F(5)$ as shown in Figure 4.14 aims at constructing a service that offers non-deterministic τ choices to each service in the set that is given.

Assumption: Let $m, n \in \mathbb{N}$ and $m, n \geq 1$. Let S and R be two interface equivalent service automata such that :

- S contains four states s_1, s_2, s_3, and s_4 with the three transitions $s_1 \xrightarrow{\tau} s_2$, $s_1 \xrightarrow{\tau} s_3$, and $s_1 \xrightarrow{\tau} s_4$ where $enable(s_2) = \{x_1, \ldots, x_m\}$, $enable(s_3) = \{x_1, \ldots, x_m, y_1, \ldots, y_n\}$, and $enable(s_4) = \{y_1, \ldots, y_n\}$,
- R contains three states s_1, s_2, and s_2 with the two transitions $s_1 \xrightarrow{\tau} s_2$ and $s_1 \xrightarrow{\tau} s_4$ where $enable(s_2) = \{x_1, \ldots, x_m\}$ and $enable(s_4) = \{y_1, \ldots, y_n\}$.

Application: There are two possible applications for Rule $F(5)$; either

1. transform R into S by inserting state s_3 and the transition $s_1 \xrightarrow{\tau} s_3$ together with a set of transitions in which each is labeled by $e \in \{y_1, \ldots, y_n\}$ and share a destination state with the each transition from either state s_2 or s_4 with the same label; or
2. transform S into R by removing state s_3 and all its adjacent transitions.

The following lemma justifies rule $F(5)$.

Lemma 4.79 (Justification of rule $F(5)$).
For each two interface equivalent services S and R that are related by Rule $F(5)$:

$$S =_{SF} R \text{ and } S =_{RF} R.$$

Proof. Clearly, services S and R are trace equivalent and the set X of events that is refused by S after performing σ is also refused by R after performing σ and vice versa.

$S =_{SF} R$: The refusal set at state s_3 refuses a subset of the refusal set at state s_2 and at state s_4. Property 4.23 illustrates that, if a service can refuse the set X of events after performing σ, then a service can also refuse any subset Y of X after performing σ. This means, $S =_{SF} R$ holds.

$S =_{RF} R$: The refusal set at the stable τ-strongly connected component containing state s_3 refuses a subset of the refusal set at the stable τ-strongly connected component

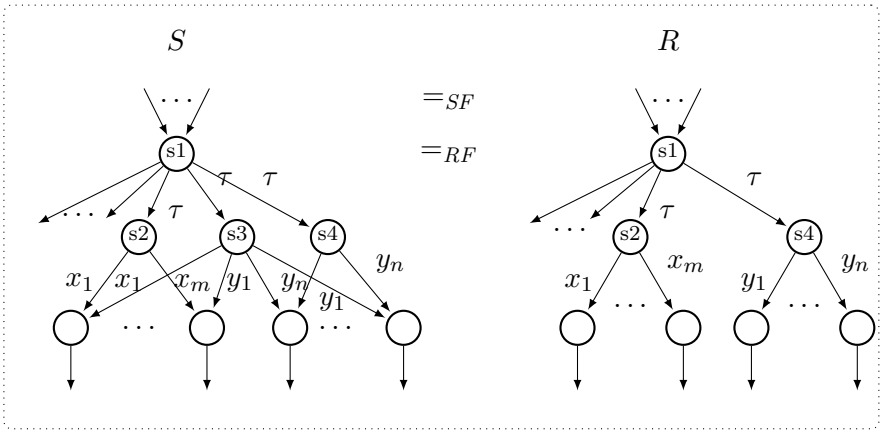


Figure 4.14.: Transformation rule $F(5)$ that preserves refinement of both stable failures and responsive failures.

containing state s_2 and the one containing state s_4. Property 4.44 illustrates that, if a service can refuse the set X of events after performing σ, then a service can also refuse any subset Y of X after performing σ. This means, $S =_{RF} R$ holds.

Thus, $S =_{SF} R$ and $S =_{RF} R$ hold. □

Next, we define rule $F(6)$ as follows.

Definition 4.80 (Rule $F(6)$).
Rule $F(6)$ as shown in Figure 4.15 aims at constructing a service that offers non-deterministic τ choices to each service in the set that is given.

Assumption: Let $n \in \mathbb{N}$ and $n > 1$. Let $\mathcal{S}$ be the set of n interface equivalent services, i. e., $\mathcal{S} = \{S_1, \ldots, S_n\}$.

Application: Transform the set $\mathcal{S}$ of interface equivalent services into service $sum(\mathcal{S})$ in which $sum(\mathcal{S})$ offers a non-deterministic τ-alternative from its initial state q_0 to each initial state q_i of each service $S_i \in sum(\mathcal{S})$.

The following lemma justifies rule $F(6)$.

Lemma 4.81 (Justification of rule $F(6)$).
For each $k \in \mathbb{N}$, each compatibility criterion $B \in \{df, rp\}$, and each two interface equivalent services S and T that are related by Rule $F(6)$:

$$(\forall S_i \in \mathcal{S} :: \ sum(\mathcal{S}) \ \sqsubseteq_{SF} \ S_i \ \text{ and } \ sum(\mathcal{S}) \ \sqsubseteq_{RF} \ S_i).$$

Proof. Consider $S_i \in \mathcal{S}$.

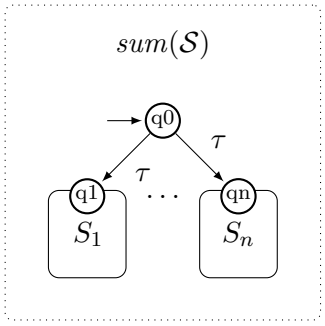


Figure 4.15.: A service $sum(\mathcal{S})$ offers a non-deterministic τ alternative to all interface equivalent service automata in the set $\mathcal{S} = \{S_1, \ldots, S_n\}$ with $n \in \mathbb{N}^+$.

$sum(\mathcal{S}) \sqsubseteq_{SF} S_i$: clearly, every stable failure in S_i is also a stable failure of $sum(\mathcal{S})$, as $sum(\mathcal{S})$ offers a non-deterministic τ path to S_i.

$sum(\mathcal{S}) \sqsubseteq_{RF} S_i$: clearly, every responsive failure in S_i is also a responsive failure of $sum(\mathcal{S})$, as $sum(\mathcal{S})$ offers a non-deterministic τ path to S_i.

Thus, service S_i refines $sum(\mathcal{S})$ under both stable failures and responsive failures for each $S_i \in \mathcal{S}$. □

Given a set $\mathcal{S}$ of services, rule $F(6)$ allows combine all services in the set $\mathcal{S}$ into one single service $sum(\mathcal{S})$ that is a substitute for every single service in the set $\mathcal{S}$. One useful application of $F(6)$, synthesizing a service that preserves selected partners, will be discussed in Section 7.3.4.

Example 4.82. Figure 4.16 shows different orders of applying various transformation rules, each preserves refinement of responsive failures. Service M on the left-most hand side can be transformed eventually into service N on the right-most by applying a number of responsive-preserving transformation rules. The application of these rules guarantees that N refines M under both stable failures and responsive failures.

Note that it is not possible to apply rule $RF(1)$ to transform R into N by removing the left branch that leads to a state with enabled events x and *final* but retaining the right branch that leads to a state with enabled events y and τ. By assumption, rule $RF(1)$ forbids to do so on the right branch of R. ◁

4.5.4. Deadlock-freedom and Responsiveness Transformation Rules

In this section, we present two transformation rules $R(1)$ and $R(2)$, each preserves both k-deadlock freedom equivalence and k-responsiveness equivalence for message bound $k \in \mathbb{N}$ on each message channel. The rule $R(1)$ is established on a nature of an asynchronous communication model where it is always possible to execute a message sending event $!e$ as long as the composition of services does not violated the k-boundedness. The rule $R(2)$ is established on a nature of a final state that enables an internal τ event.

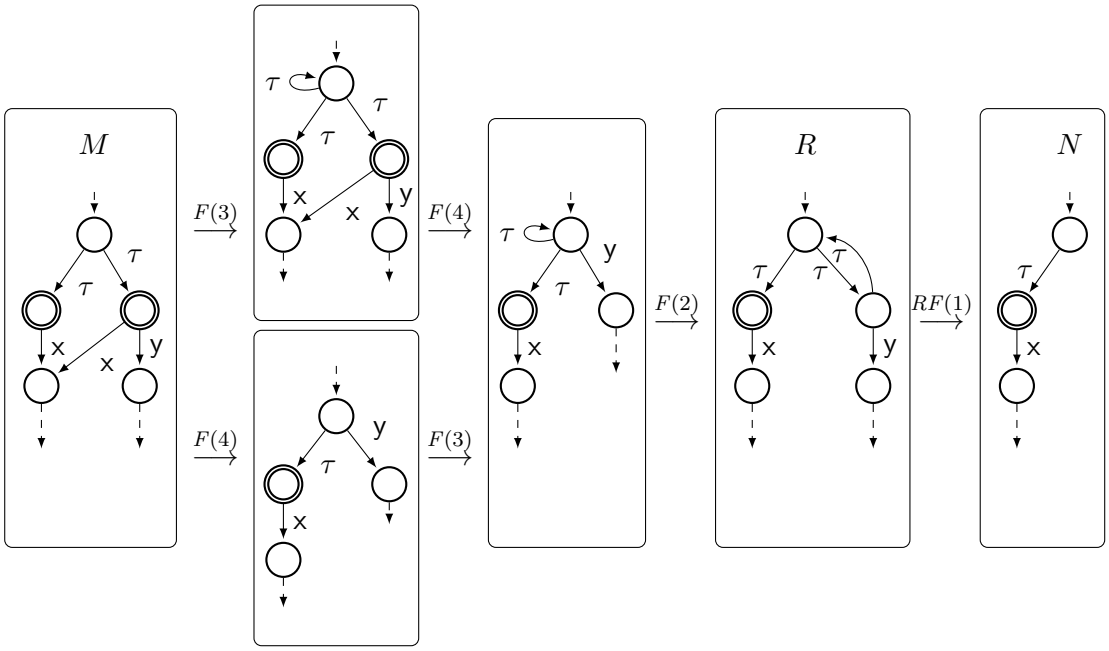


Figure 4.16.: Fragment of service N in which service N can be transformed from service M by applying a number of transformation rules.

First, we define rule $R(1)$ as follows.

Definition 4.83 (Rule $R(1)$).
Rule $R(1)$ as shown in Figure 4.17 aims at either removing or inserting a τ transition that reaches a choice containing message sending events.

Assumption: Let S and R be two interface equivalence service automata such that :

- S contains state s_1 and a transition $s_1 \xrightarrow{\tau} s_2$ with $enable(s_1) \neq \emptyset$ and $enable(s_2) \subseteq\ !\Sigma$, and
- R contains a state r_1 with $enable(r_1) \neq \emptyset$ and $enable(r_1) \subseteq !\Sigma$.

Application(s): There are two possible applications for Rule $R(1)$; either

1. transform R into S by replacing state r_1 by the transition $s_1 \xrightarrow{\tau} s_2$; or
2. transform S into R by replacing states s_1, and transition $s_1 \xrightarrow{\tau} s_2$ by state r_1;

such that $incoming(r_1) = incoming(s_1)$ and $enable(s_1) \cup enable(s_2) = enable(r1) \cup \{\tau\}$ must hold for each application.

The following lemma justifies rule $R(1)$.

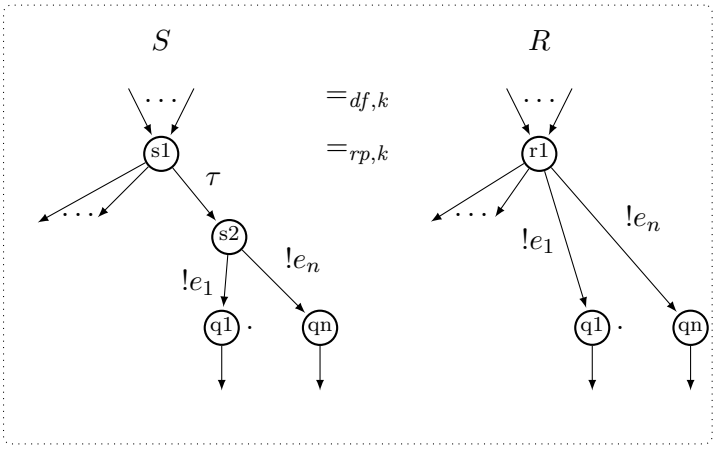


Figure 4.17.: Transformation rule $R(1)$ that preserves both k-deadlock freedom equivalence and k-responsiveness equivalence for each message bound $k \in \mathbb{N}^+$ on each message channel.

Lemma 4.84 (Justification of rule $R(1)$).
For each message bound $k \in \mathbb{N}$ and each two interface equivalent services S and R that are related by Rule $R(1)$:

$$S \ =_{df,k} \ R \ \text{ and } \ S \ =_{rp,k} \ R.$$

Proof. Let P be a service with compatible interface to S and R.

$S \sqsubseteq_{df,k} R$: Suppose $S \oplus R$ can reach a deadlock state q which is not a final state and has no outgoing transition. Such state q does not involve state r_1 for $n \geq 1$, as R can always perform $!e_i$ for $1 \leq i \leq n$ at r_1 and reach state q_i. The composition $S \oplus P$ can also reach such state q.

$S \sqsupseteq_{df,k} R$: Suppose $S \oplus P$ can reach a deadlock state q which is not a final state and has no outgoing transition. Such state q does not involve state s_2, as S can always perform $!e_i$ for $1 \leq i \leq n$ at s_2 and reach state q_i. The composition $S \oplus R$ can also reach such state q.

$S \sqsubseteq_{rp,k} R$: Suppose $S \oplus R$ can reach a divergent state q that excludes a trace from R. Such state q does not involve state r_1 for $n \geq 1$, as R can always perform $!e_i$ for $1 \leq i \leq n$ at ${}_r1$ and reach state q_i. The composition $S \oplus P$ can also reach such state q.

$S \sqsupseteq_{rp,k} R$: Suppose $S \oplus P$ can reach a divergent state q that excludes a trace from S. Such state q does not involve state s_2, as S can always perform $!e_i$ for $1 \leq i \leq n$ at s_2 and reach state q_i. The composition $S \oplus R$ can also reach such state q.

Thus, $S =_{df,k} R$ and $S =_{rp,k} R$ holds. □

Next, we define rule $R(2)$ as follows.

Definition 4.85 (Rule $R(2)$).
Rule $R(2)$ as shown in Figure 4.18 aims at either removing or inserting a terminating event *final* at a state where a τ event is enabled.

Assumption: Let S and R be two interface equivalent service automata such that :

- S contains a final state s_1 with $\tau, \mathit{final} \in \mathit{enable}(s_1)$ and s_1 is not in a stable τ-strongly connected component of S,
- R contains a non-final state r_1 with $\tau \in \mathit{enable}(r_1)$ and r_1 is not in a stable τ-strongly connected component of R.

Application(s): There are two possible applications for Rule $R(1)$; either

1. transform S into R by replacing a final state s_1 with a non-final state r_1; or
2. transform R into S by replacing a non-final state r_1 with a final state s_1;

such that $\mathit{incoming}(r_1) = \mathit{incoming}(s_1)$ and $\mathit{enable}(s_1) = \mathit{enable}(r1) \cup \{\mathit{final}\}$ must hold for each application.

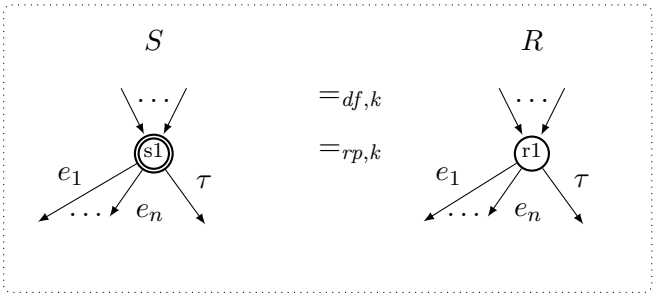


Figure 4.18.: Transformation rule $R(2)$; that preserves both k-deadlock freedom equivalence and k-responsiveness equivalence for each message bound $k \in \mathbb{N}^+$ on each message channel.

The following lemma justifies rule $R(2)$.

Lemma 4.86 (Justification of rule $R(2)$).
For each message bound $k \in \mathbb{N}$ and each two interface equivalent services S and R that are related by Rule $R(2)$:

$$S =_{df,k} R \text{ and } S =_{rp,k} R.$$

Proof. Let P be a service with compatible interface to S and R.

$S \sqsubseteq_{df,k} R$: Suppose $S \oplus R$ can reach a deadlock state q which is not a final state and no outgoing transition. Such state q does not involve state r_1, as R can always perform τ. The composition $S \oplus P$ can also reach such state q.

$S \sqsupseteq_{df,k} R$: Suppose $S \oplus P$ can reach a deadlock state q which is not a final state and no outgoing transition. Such state q does not involve state s_1, as S can always perform τ. The composition $S \oplus R$ can also reach such state q.

$S \sqsubseteq_{rp,k} R$: Suppose $S \oplus R$ can reach a divergent state q that excludes a trace from R. Such state q does not involve state r_1, as r_1 is not in a stable τ-strongly connected component. The composition $S \oplus P$ can also reach such state q.

$S \sqsupseteq_{rp,k} R$: Suppose $S \oplus P$ can reach a divergent state q that excludes a trace from S. Such state q does not involve state s_1, as s_1 not in a stable τ-strongly connected component. The composition $S \oplus R$ can also reach such state q.

Thus, $S =_{df,k} R$ and $S =_{rp,k} R$ holds. □

We will apply the two rules $R(1)$ and $R(2)$ in Section 7.3.3 to construct a minimal public view for a service.

4.6. Concluding Remarks

In this chapter, we introduced a canonical controller as a canonical representative of all controllers of a given service. A canonical controller of a given service is a controller with distinguished properties and therefore can be regarded as a *behavioral specification* of all controllers of a given service. For checking whether an implementation conforms to the specification (i. e., whether another service is a controller of a given service), we apply different refinement relation according to the given behavioral compatibility criterion as a conformance relation between implementation and the specification.

In case of deadlock freedom, we identify the stable failures refinement (e. g., in the sense of Roscoe [1998], Brookes et al. [1984], Hoare [1978, 1985b] defined for labeled transition systems) as the conformance relation between a deadlock-free controller and a canonical deadlock-free controller. Though the stable failures refinement is finer than deadlock freedom preorder, we show that a canonical deadlock-free controller is in the stable failures preorder larger than or equal to every other deadlock-free controller.

In case of responsiveness, none of the refinement relations known to us is a good candidate for relating a responsive controller with a canonical responsive controller. For this purpose, we propose an extension of stable failure semantics to *responsive failures* semantics. We show that the responsive failures refinement is finer than responsiveness preorder; nevertheless, a canonical responsive controller is in the responsive failures preorder larger than or equal to every other responsive controller of a given service. Consequently, we identify the responsive failures refinement as the conformance relation between a responsive controller and a canonical responsive controller.

In the line of a conformance relation with respect to deadlock freedom, Stahl and Vogler [2011] have investigated a trace-based view on the operating guidelines which induces a conformance relation between two substitution services. Kaschner [2011] has studied conformance testing for asynchronously communicating services based on operating guidelines approach. We refer also to Section 2.5 for a survey on related work.

For a given service, its canonical controller can be constructed from its operating guideline by replacing each of its states with a fragment of nondeterministic internal events between all choices described at the respective state of the operating guideline. In comparison to a canonical controller, an operating guideline representation is more compact as it requires less storage capacity (e. g.., for storing an operating guideline in a service repository) and yields higher efficiency for deciding a controllers [Lohmann and Wolf, 2011b]. Nevertheless, the underlying structure of an operating guideline is not a service that can canonically represent the set of all controllers. Therefore, an operation on service is not directly applicable to an operating guideline. Though its underlying service is a most permissive controller of the given service and is able to simulate every other controller of the given service, not every service simulated by the most permissive controller is a controller of the given service. We refer to Section 3.3 for related issues on operating guidelines and a most permissive controller of a given service.

In contrast to a most permissive controller, a canonical controller is a service that encodes within its structure all possible traces and choices of every controller. This makes it possible to synthesize from the constructed canonical controller an arbitrary controller by means of transformation. For this purpose, we propose a set of transformation rules each of which preserves either refinement or equivalence of both stable failures and responsive failures. Some rules can be seen as an extension of the transformation rules from the literature (e. g., van der Aalst and Basten [2002], van der Aalst et al. [2008], König et al. [2008], van der Aalst et al. [2009], Stahl [2009]). An application of these rules to the canonical controller guarantees that the transformed service is a controller of the given service. Nevertheless, the set of transformation rules presented in this chapter is restricted and not complete; this means, there is no guarantee that all controllers can be synthesized from an arbitrary controller by means of transformation.

With its relatively liberal structure, the constructed canonical controller grows exponentially in size with respect to the increased size of the most permissive controller. In this chapter, we proposed an alternative construction procedure of a canonical controller, called a compact canonical controller. The compact canonical controller can be constructed from the operating guideline by replacing each of its states with a fragment of of nondeterministic internal events between the minimal choices of the operating guideline. We illustrated that the compact canonical controller is relatively smaller in size in comparison to the (complete) canonical controller, yet, both of them are equivalent under respective failures. We will discuss their applications further in Chapter 7.

In the following chapter, we will investigate further operations on the canonical controller. We will show how to employ the various operations on the canonical controller as the closure operator of the given service as well as how to characterize the set of all substitutes for the given service under the respective behavioral compatibility criterion.

5. Canonical Representative of Inclusion Substitutes

In this chapter, we study a *maximal controller* and a *minimal controller* of a given service. We prove several useful properties of the maximal controller by establishing a relationship between the set of all substitutes for a given service under inclusion and the set of all its controllers via a maximal controller. For each behavioral compatibility criterion, we propose two construction procedures of a maximal controller of a given service. We employ a minimal controller to illustrate a significant difference between the two behavioral compatibility criteria investigated in this thesis.

The chapter is organized as follows. In Section 5.1, we define a maximal controller for a given service and prove its properties. In Section 5.2, we illustrate how to employ a maximal controller to represent the set of all services that can substitute the given service under inclusion. In Section 5.3, we study a minimal controller for each behavior compatibility criterion. Finally, Section 5.4 concludes the chapter.

5.1. Maximal Controllers

In this section, we introduce a maximal $\mathcal{B}$-controller for a given service and study how a maximal controller behaves with respect to the $\mathcal{B}$-preorder relation introduced in Chapter 2. The concept of a *maximal controller* for deadlock freedom inclusion (also called *accordance* in Mooij and Voorhoeve [2009], Stahl et al. [2009], Stahl and Wolf [2009], Mooij et al. [2011]) has been introduced in Mooij and Voorhoeve [2009] and extensively studied in Mooij, Parnjai, Stahl, and Voorhoeve [2011]. For a message bound $k \in \mathbb{N}$, the maximal $\mathcal{B}$-controller of a given service S is a distinguished $\mathcal{B}$-controller that is larger than or equal to other controllers of service S in the $\mathcal{B}$-preorder.

For a given behavioral compatibility property $\mathcal{B}$ and each k-controllable service S, we define a maximal $\mathcal{B}$-controller of S as follows.

Definition 5.1 (Maximal $\mathcal{B}$-controller).
Let $\mathcal{B} \in \{df_k, rp_k\}$ be a behavioral compatibility criterion for each message bound $k \in \mathbb{N}$. Let S be a $\mathcal{B}$-controllable service. A service $\mathcal{B}max(S)$ is a maximal $\mathcal{B}$-controller of service S iff

$$(\forall P \in \mathcal{B}\text{-}Controllers(S) : \; P \; \sqsubseteq_{\mathcal{B}} \; \mathcal{B}max(S)).$$

By definition, each $\mathcal{B}$-controller P of service S can substitute service $\mathcal{B}max(S)$ under $\mathcal{B}$ inclusion. This means that every $\mathcal{B}$-controller of $\mathcal{B}max(S)$ is also a $\mathcal{B}$-controller of P. As the $\mathcal{B}$ inclusion relation $\sqsubseteq_{\mathcal{B}}$ is a preorder relation, $\mathcal{B}max(S)$ is in the $\sqsubseteq_{\mathcal{B}}$ preorder larger than or equal to any other element P of the set $\mathcal{B}\text{-}Controllers(S)$ of all $\mathcal{B}$-controllers of S. Therefore, $\mathcal{B}max(S)$ is a maximal element of the set $(\mathcal{B}\text{-}Controllers(S), \sqsubseteq_B)$ that is equipped with $\sqsubseteq_{\mathcal{B}}$ relation.

For a given service S, all $\mathcal{B}$-controllers of $\mathcal{B}max(S)$ are included in the set of all $\mathcal{B}$-controllers of a service P that is a controller of S. Thus, the set of all $\mathcal{B}$-controllers of $\mathcal{B}max(S)$ can be formed by intersecting the sets of $\mathcal{B}$-controllers of each $\mathcal{B}$-controller P_i of S, as illustrated in Figure 5.1.

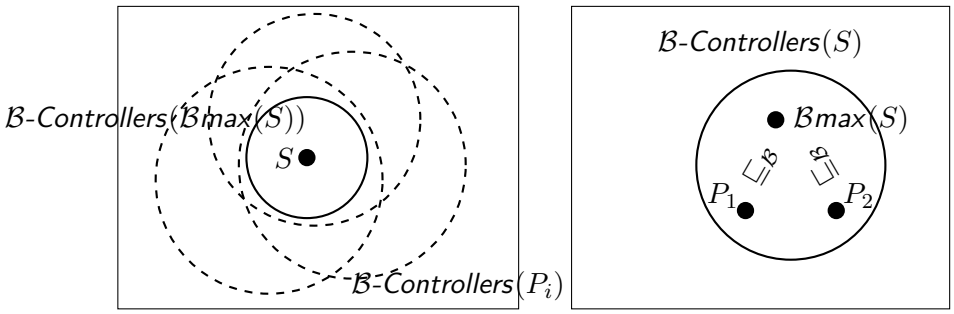


Figure 5.1.: The intersection of all $\mathcal{B}$-controllers P_i that forms the set of all $\mathcal{B}$-controllers of $\mathcal{B}max(S)$.

The maximal $\mathcal{B}$-controller is unique up to $=_{\mathcal{B}}$ for $\mathcal{B} \in \{df_k, rp_k\}$ and a message bound $k \in \mathbb{N}$. This implies that possibly there exists more than one maximal controller for a given service S. It is highly important in practice that there exists has st least one maximal $\mathcal{B}$-controller $\mathcal{B}max(S)$ of service S.

In the remainder of this section, we fix one maximal $\mathcal{B}$-controller $\mathcal{B}max(S)$ of a given service S. We illustrate in Section 5.1.1 the properties of a maximal controller that is independent of the specific behavioral compatibility criterion $\mathcal{B}$. In Section 5.1.2, we show how to construct two distinguished maximal k-deadlock-free controllers for deadlock freedom ($\mathcal{B} = df_k$). In Section 5.1.3, we show how to construct two distinguished maximal k-responsive controllers for responsiveness ($\mathcal{B} = rp_k$). Note, that the latter two sections share common techniques to construct a maximal controller, and they employ the properties studied in Section 5.1.1 for the construction. Though it is useful to study the differences.

5.1.1. Properties of Maximal Controllers

In this section, we study how a maximal $\mathcal{B}$-controller behaves with respect to a $\mathcal{B}$-preorder relation. For this purpose, we consider two sets of services related to a given service S; these two sets are the set $\mathcal{B}$-*Inclusion*(S) of all services that can substitute S under $\mathcal{B}$ inclusion and the set $\mathcal{B}$-*Controllers*(S) of all $\mathcal{B}$-controllers of S. We equip each set with the same $\sqsubseteq_{\mathcal{B}}$ preorder relation, and establish a *Galois connection* [see e. g., Ore, 1944, Backhouse, 2002] of the $\mathcal{B}$-preordered set $(\mathcal{B}\text{-}Inclusion(S), \sqsubseteq_{\mathcal{B}})$ and the $\mathcal{B}$-preordered set $(\mathcal{B}\text{-}Controllers(S), \sqsubseteq_{\mathcal{B}})$ via a maximal $\mathcal{B}$-controller $\mathcal{B}max(S)$ of service S.

[Mooij and Voorhoeve, 2009] have illustrated that the maximal controller function $\mathcal{B}max$ inverts the $\sqsubseteq_{\mathcal{B}}$ preorder on the set of $\mathcal{B}$-controllable services.

Lemma 5.2 ($\mathcal{B}max$ inverts $\mathcal{B}$-preorder). [Mooij and Voorhoeve, 2009]
For each message bound $k \in \mathbb{N}$, each compatibility criterion $\mathcal{B} \in \{df_k, rp_k\}$, and each two $\mathcal{B}$-controllable services S and P with compatible interface:

$$S \sqsubseteq_{\mathcal{B}} \mathcal{B}max(P) \quad \Leftrightarrow \quad P \sqsubseteq_{\mathcal{B}} \mathcal{B}max(S).$$

Proof. Given:

$$\begin{aligned} P \sqsubseteq_{\mathcal{B}} \mathcal{B}max(S) \;&\Leftrightarrow\; \{\text{Definition 5.1}\} \\ &\quad\; P \in \mathcal{B}\text{-}Controllers(S) \\ &\Leftrightarrow\; \{\text{Property 2.27; } \mathcal{B}\text{-controller relation is symmetric}\} \\ &\quad\; S \in \mathcal{B}\text{-}Controllers(P) \\ &\Leftrightarrow\; \{\text{Definition 5.1}\} \\ &\quad\; S \sqsubseteq_{\mathcal{B}} \mathcal{B}max(P) \end{aligned}$$

Consequently, the lemma holds. □

Based on Lemma 5.2, we obtain a stronger anti-monotonicity property of $\mathcal{B}max$.

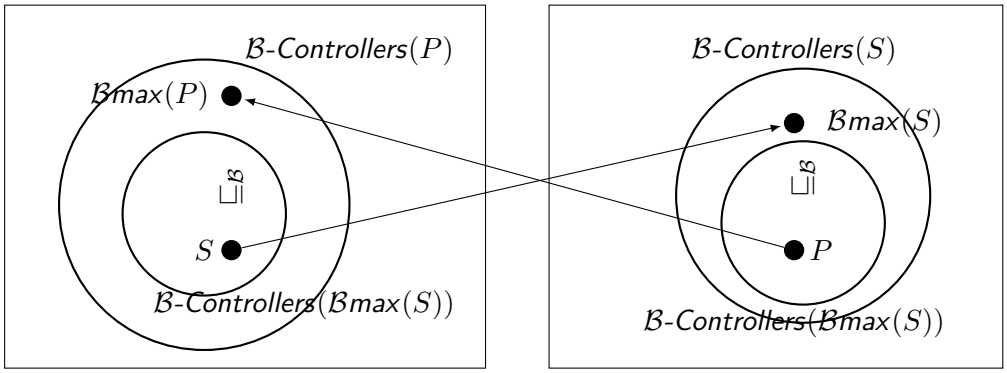


Figure 5.2.: Illustration of Lemma 5.2 states that the function $\mathcal{B}max$ inverts the B-preorder $\sqsubseteq_{\mathcal{B}}$ on sets of B-controllable services.

Lemma 5.3 (Anti-monotonicity of $\mathcal{B}max$).
For each message bound $k \in \mathbb{N}$, each compatibility criterion $\mathcal{B} \in \{df_k, rp_k\}$, and each two $\mathcal{B}$-controllable and services S and T with equivalent interface:

$$T \sqsubseteq_{\mathcal{B}} S \Leftrightarrow \mathcal{B}max(S) \sqsubseteq_{\mathcal{B}} \mathcal{B}max(T).$$

Proof. We calculate :

$$\begin{aligned} T \sqsubseteq_{\mathcal{B}} S \Leftrightarrow\ & \{\text{definition of } \sqsubseteq_{\mathcal{B}}\} \\ & \mathcal{B}\text{-}Controllers(T) \supseteq \mathcal{B}\text{-}Controllers(S) \\ \Leftrightarrow\ & \{\text{Definition 5.1 (twice); set theory}\} \\ & (\forall P :: P \sqsubseteq_{\mathcal{B}} \mathcal{B}max(S) \Rightarrow P \sqsubseteq_{\mathcal{B}} \mathcal{B}max(T)) \\ \Leftrightarrow\ & \{\text{indirect equality}\} \\ & \mathcal{B}max(S) \sqsubseteq_{\mathcal{B}} \mathcal{B}max(T) \end{aligned}$$

Consequently, the lemma holds. □

Based on Lemma 5.2 and Lemma 5.3, we obtain that the function composition $\mathcal{B}max^2 = (\mathcal{B}max \circ \mathcal{B}max)$ is the identity with respect to the $\mathcal{B}$-equivalence $=_{\mathcal{B}}$.

Theorem 5.4 (Closure operation of services).
For each message bound $k \in \mathbb{N}$, each compatibility criterion $\mathcal{B} \in \{df_k, rp_k\}$, and each $\mathcal{B}$-controllable service S :

$$S =_{\mathcal{B}} \mathcal{B}max(\mathcal{B}max(S)).$$

Proof. We calculate :

$$
\begin{aligned}
S =_{\mathcal{B}} \mathcal{B}max(\mathcal{B}max(S)) \;\Leftrightarrow\; & \{\text{indirect equality}\} \\
& (\forall T ::\; T \sqsubseteq_{\mathcal{B}} S \;\Leftrightarrow\; T \sqsubseteq_{\mathcal{B}} \mathcal{B}max(\mathcal{B}max(S))) \\
\Leftrightarrow\; & \{\text{Lemma 5.2}\} \\
& (\forall T ::\; T \sqsubseteq_{\mathcal{B}} S \;\Leftrightarrow\; \mathcal{B}max(S) \sqsubseteq_{\mathcal{B}} \mathcal{B}max(T)) \\
\Leftrightarrow\; & \{\text{Lemma 5.3}\} \\
& \mathit{true}
\end{aligned}
$$

Consequently, the theorem holds. □

Theorem 5.4 illustrates a relationship between the two sets of services that are related to a given service S. These two sets are the set ($\mathcal{B}$-*Controllers*(S), $\sqsubseteq_{\mathcal{B}}$) and the set ($\mathcal{B}$-*Inclusion*($S$), $\sqsubseteq_{\mathcal{B}}$), each equipped the same $\sqsubseteq_{\mathcal{B}}$ preorder relation. The relationship between the two sets can be established via function $\mathcal{B}max$ and function composition $\mathcal{B}max^2 = (\mathcal{B}max \circ \mathcal{B}max)$.

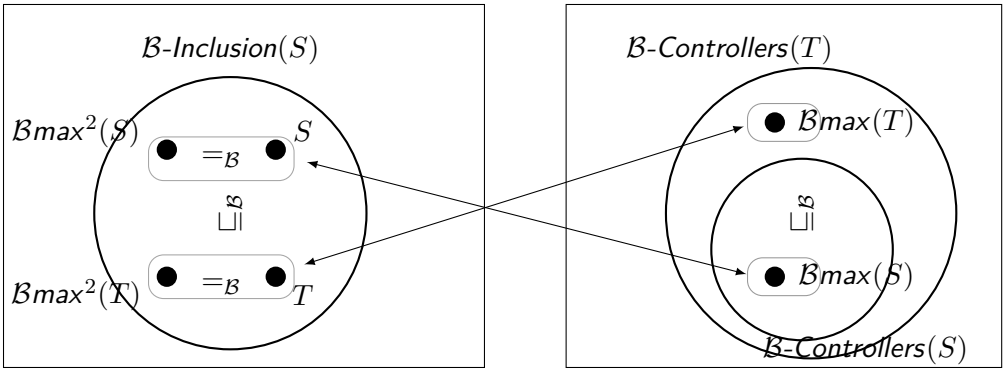


Figure 5.3.: Illustration of Lemma 5.3 and Theorem 5.4; where $\mathcal{B}max$ is the anti-monotonicity function on sets of services and $\mathcal{B}max^2 = (\mathcal{B}max \circ \mathcal{B}max)$ is the function composition that is a closure operator on sets of services.

Consider the function composition $\mathcal{B}max^2 = (\mathcal{B}max \circ \mathcal{B}max)$. In the following property, we illustrate that $\mathcal{B}max^2$ is indeed a closure operator on the set of $\mathcal{B}$-controllable services.

Property 5.5 For each message bound $k \in \mathbb{N}$, each compatibility criterion $\mathcal{B} \in \{df_k, rp_k\}$, and each $\mathcal{B}$-controllable service S:

(a) $\mathcal{B}max^2$ is *extensive* : $\forall S :: S \;\sqsubseteq_{\mathcal{B}}\; \mathcal{B}max^2(S)$;

(b) $\mathcal{B}max^2$ is *monotonic* : $\forall T :: T \sqsubseteq_{\mathcal{B}} S \;\Rightarrow\; \mathcal{B}max^2(T) \sqsubseteq_{\mathcal{B}} \mathcal{B}max^2(S)$; and

(c) $\mathcal{B}max^2$ is *idempotent* : $\forall S :: \mathcal{B}max^2(\mathcal{B}max^2(S)) \;=_{\mathcal{B}}\; \mathcal{B}max^2(S)$.

Extensiveness of $\mathcal{B}max^2$ means that the set $\mathcal{B}$-*Inclusion*$(\mathcal{B}max^2(S))$ contains S. Monotonicity of $\mathcal{B}max^2$ means that if T is contained in the set $\mathcal{B}$-*Inclusion*(S), then also $\mathcal{B}max^2(T)$ is contained in the set $\mathcal{B}$-*Inclusion*$(\mathcal{B}max^2(S))$. Idempotency of $\mathcal{B}max^2$ means that $\mathcal{B}max^2$ (of S) and $\mathcal{B}max^2$ (of $\mathcal{B}max^2$ (of S)) are equivalent under $\mathcal{B}$. This implies that the closure $\mathcal{B}max^2$ can be applied multiple times without changing the result beyond the initial application.

These properties of a maximal $\mathcal{B}$-controller demonstrate the value of Theorem 5.4 as the fixed point equation $S =_{\mathcal{B}} \mathcal{B}max^2(S)$, where we can express the *specification* of services directly in terms of a solution of the fixed point equation.

Based on Lemma 5.3, we obtain a procedure for deciding $\mathcal{B}$ inclusion. For each two $\mathcal{B}$-controllable services S and T, whether service T can substitute service S under $\mathcal{B}$ inclusion can be decided by checking if the composition of T and a maximal $\mathcal{B}$-controller $\mathcal{B}max(S)$ of S satisfies the compatibility criterion $\mathcal{B}$.

Theorem 5.6 (Composition with $\mathcal{B}max$ decides $\mathcal{B}$ inclusion).
For each message bound $k \in \mathbb{N}$, each compatibility criterion $\mathcal{B} \in \{df_k, rp_k\}$, and each two $\mathcal{B}$-controllable and interface equivalent services S and T:

$$T \sqsubseteq_{\mathcal{B}} S \quad \Leftrightarrow \quad \mathcal{B}(T \oplus \mathcal{B}max(S)).$$

Proof. We calculate :

$$\begin{aligned}
\mathcal{B}(T \oplus \mathcal{B}max(S)) \;&\Leftrightarrow\; \{\text{Definition of } \mathcal{B}\text{-Controller}\} \\
&\quad\; \mathcal{B}max(S) \in \mathcal{B}\text{-}Controllers(T) \\
&\Leftrightarrow\; \{\text{Definition 5.1}\} \\
&\quad\; \mathcal{B}max(S) \;\sqsubseteq_{\mathcal{B}}\; \mathcal{B}max(T) \\
&\Leftrightarrow\; \{\text{Lemma 5.3}\} \\
&\quad\; T \;\sqsubseteq_{\mathcal{B}}\; S
\end{aligned}$$

Consequently, the theorem holds. □

Similar decision procedure for different preorder relations have been studied. For instance, Dill [1989, 1990], Lin et al. [1995a,b], Zhou et al. [2000] have considered the *conformance preorder* between specification $\mathcal{S}$ and implementation $\mathcal{I}$, where implementation $\mathcal{I}$ can substitute the specification $\mathcal{S}$ in any possible environment if the composition of implementation $\mathcal{I}$ and the *mirror* of specification $\mathcal{S}$ yields a failure-free result. We refer to Section 2.5 for more discussion on the related preorder, to Section 4.3 and Section 4.4 for the relationship between our preorder and classical failures preorder, and to Section 7.1 for an analysis of our decision procedure.

Based on Theorem 5.6, we obtain the characterization of the set $\mathcal{B}$-*Inclusion*(S) of all substitution services of S under $\mathcal{B}$ inclusion. With the following theorem, we can characterize $\mathcal{B}$-*Inclusion*(S) in terms of the set $\mathcal{B}$-*Controllers*$(\mathcal{B}max(S))$ of all $\mathcal{B}$-controllers of a maximal $\mathcal{B}$-controller of a $\mathcal{B}$-controllable service S.

Theorem 5.7 (Characterization of substitutes under inclusion).
For each message bound $k \in \mathbb{N}$, each compatibility criterion $\mathcal{B} \in \{df_k, rp_k\}$, and each $\mathcal{B}$-controllable services S:

$$\mathcal{B}\text{-}Inclusion(S) \;=\; \mathcal{B}\text{-}Controllers(\mathcal{B}max(S)).$$

Proof. Let T be an interface equivalent service to S.

$$\begin{aligned} T \in \mathcal{B}\text{-}Controllers(\mathcal{B}max(S)) \;&\Leftrightarrow\; \{\text{Definition of } \mathcal{B}\text{-Controller}\} \\ &\quad\; \mathcal{B}(T \oplus \mathcal{B}max(S)) \\ &\Leftrightarrow\; \{\text{Theorem 5.6; Definition of B-preserving preorder}\} \\ &\quad\; T \in \mathcal{B}\text{-}Inclusion(S) \end{aligned}$$

Consequently, the theorem holds. □

The theorem illustrates that we can reduce the characterization problem of all substitutes for a given service S under $\mathcal{B}$ inclusion to the characterization of all $\mathcal{B}$-controllers of a maximal $\mathcal{B}$-controller of S.

In case we can synthesize a maximal $\mathcal{B}$-controller from a given service S, it is possible to realize Theorem 5.7 by employing the operating guidelines technique as introduced in Section 3.3 to construct a finite representation of all controllers of the synthesize controller. We refer to Section 5.2.1 for a related discussion on this issue.

Intuitively, the characterization of all $\mathcal{B}$ inclusion services is also a natural candidate for a decision procedure for $\mathcal{B}$ inclusion. We refer to Section 7.1 for an analysis of various decision procedures. This characterization all $\mathcal{B}$ inclusion services also enabled other useful applications, we refer to Section 7.4 for more on the issues regarding correcting undesirable service behavior and improving service behavior.

In this section, we present a maximal $\mathcal{B}$-controller and prove some of its useful properties which illustrates how it behaves with respect to the $\sqsubseteq_{\mathcal{B}}$ preorder relation. To realize the properties of a maximal $\mathcal{B}$-controller of a given service S, it is required that the function $\mathcal{B}max$ has at least one solution. We illustrate a construction procedure of two distinguished maximal k-deadlock-free controllers of a given service in Section 5.1.2 and a construction procedure of two distinguished maximal k-responsive controllers of a given service in Section 5.1.3. We discuss their related applicability and experimental results in Chapter 7.

5.1.2. Constructing a Maximal Deadlock-free Controller

In this section, we consider k-deadlock freedom as a behavioral compatibility criterion ($\mathcal{B} = df_k$) and present construction procedures of two maximal k-deadlock-free controllers from a given service. We prove that each constructed controller is indeed a solution for a maximal k-deadlock-free controller of a given service for each message bound $k \in \mathbb{N}$.

To this respect, we first define a maximal k-deadlock-free controller of a service as an instance of Definition 5.1 for $\mathcal{B} = df_k$.

Definition 5.8 (Maximal k-deadlock-free controller).
For each message bound $k \in \mathbb{N}$, a maximal k-deadlock-free controller of a k-deadlock-free controllable service S is a service $max_{df,k}(S)$ that satisfies the following property:

$$(\forall P \in df_k Controllers(S) : P \sqsubseteq_{df,k} max_{df,k}(S)).$$

Given a k-deadlock-free controllable service S, we show with the following lemma that there exists an (infinite) maximal k-deadlock-free controller of S.

Lemma 5.9 (Existence of a maximal k-deadlock-free controller).
For each message bound $k \in \mathbb{N}$ and each k-deadlock-freely controllable service S, there exists a maximal deadlock-free controller of S.

Proof. Let $P^* = sum(df_k Controllers(S))$ be constructed from the (possibly infinite) set $df_k Controllers(S)$ of S by applying Rule $F(6)$ (Definition 4.80, Chapter 4). The justification of Rule $F(6)$ (Lemma 4.81) guarantees that for each $P \in df_k Controllers(S)$ holds: $P^* \sqsubseteq_{SF} P$. That is, each $P \in df_k Controllers(S)$ refines P^* under stable failures. As stable failures refinement implies deadlock freedom Inclusion (Lemma 4.34), it follows that P is smaller than or equal to P^* in the deadlock-free preorder, that is, $P \sqsubseteq_{df,k} P^*$ holds.

Assume P^* is not a deadlock-free controller of S, i. e., $P^* \notin df_k Controllers(S)$. By definition, the composition $S \oplus P^*$ is not k-deadlock-free, and by definition, there is a deadlock state q in $S \oplus P^*$. As S is k-deadlock-freely controllable, by definition $df_k Controllers(S) \neq \emptyset$. By construction, P^* offers a non-deterministic τ choice to all services in $df_k Controllers(S)$. As $S \oplus P^*$ contains a deadlock state by assumption, it follows that there must be a service P in the set $df_k Controllers(S)$ such that the composition $S \oplus P$ contains the deadlock state q. This means, $\neg df_k(S \oplus P)$ must hold, which contradicts to the assumption $P \in df_k Controllers(S)$.

Thus, $P^* = sum(df_k Controllers(S))$ is indeed an (infinite) maximal k-deadlock-free controller of S. □

Next, we recall a canonical k-deadlock-free controller $\mathcal{C}_{df,k}(S)$ of service S, discussed in Section 4.1, of a given service S. The controller $\mathcal{C}_{df,k}(S)$ is constructed from a finite representation of all k-deadlock-free controllers of S (known as a deadlock-free operating guideline of S) by replacing each labeled state q with a fragment of nondeterministic internal alternative between all deadlock-free choices that are described at state q.

In Theorem 5.11, we prove that the canonical k-deadlock free controller $\mathcal{C}_{df,k}(S)$ is indeed a maximal k-deadlock-free controller of service S. As $\mathcal{C}_{df,k}(S)$ is a k-deadlock-free controller of S (Corollary 4.2), we first prove in Lemma 5.10 that the controller $\mathcal{C}_{df,k}(S)$ is in the deadlock freedom preorder larger than or equal to every k-deadlock-free controller of S.

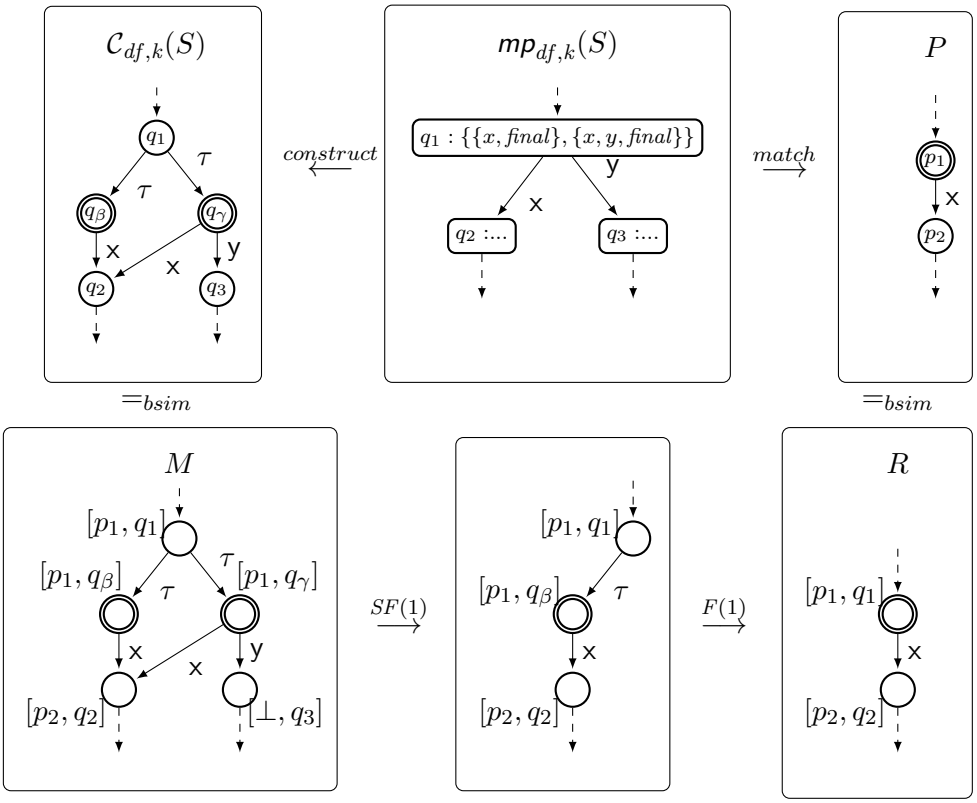


Figure 5.4.: A service R that matches with $mp_{df,k}(S)$ can be obtained from $\mathcal{C}_{df,k}(S)$.

Lemma 5.10 ($\mathcal{C}_{df,k}(S)$ is larger than or equal to a k-deadlock-free controller of S).
For each message bound $k \in \mathbb{N}$ and each k-deadlock-freely controllable service S :

$$(\forall P :: P \in df_k Controllers(S)) \;\Rightarrow\; (P \;\sqsubseteq_{df,k}\; \mathcal{C}_{df,k}(S))).$$

Proof. Let S and $P = [Q_P, q_{0P}, I, O, \rightarrow_P, \Omega_P]$ be two k-deadlock-freely controllable services with compatible interfaces. Let $mp_{df,k}(S)^\varphi = [mp_{df,k}(S), \varphi] = [Q, q_0, I, O, \rightarrow, \Omega, \varphi]$ be a k-deadlock-free operating guideline of S, and $\mathcal{C}_{df,k}(S)$ be constructed from $mp_{df,k}(S)^\varphi$ as described in Definition 4.1.

Let $R = [Q_R, q_{0R}, I, O, \rightarrow, \Omega_R]$ be a service with

- $Q_R = q_P \times Q$;
- $q_{0R} = [q_{0P}, q]$;
- $\Omega_R = \Omega_P \times Q$;
- $\rightarrow_R = \{[q_P, q] \xrightarrow{m} [q'_P, q'] \mid q_P \xrightarrow{m}_P q'_P \;\wedge\; q \xrightarrow{m} q'\}$;

Service R synchronizes P with $df\text{-}mp_k(S)$ but ignores transitions that are not shared. Each state $[p_P, q]$ in R is bisimilar to state q_P in P as $df\text{-}mp_k(S)$ simulates P and

df-mp$_k(S)$ is deterministic. See Figure 5.4.

Let $M = [Q_M, q_{0M}, I, O, \rightarrow_M, \Omega_M]$ be a service with

- $Q_M = (q_P \cup \{\bot\}) \times (Q \cup \{q_\beta \mid q \in Q \ \wedge \ \beta \in \varphi_k(q)\})$;
- $q_{0M} = [q_{0P}, q]$;
- $\Omega_M = (q_P \cup \{\bot\}) \times \{Q_\beta \mid \beta \in \varphi_k(q)\}$;
- $\rightarrow_M = \ \{[q_P, q] \xrightarrow{\tau}_M [q_P, q_\beta] \mid q \in Q \wedge \beta \in \varphi_k(q)\} \cup$
  $\{[q_P, q_\beta] \xrightarrow{m}_M [q'_P, q'] \mid q \xrightarrow{m} q' \ \wedge \ \beta \in \varphi_k(q) \ \wedge$
  $(q_P \xrightarrow{m}_P q'_P \vee (q'_P = \bot \ \wedge \neg(\exists q''_P :: q_P \xrightarrow{m}_P q''_P)))\}$

Service M synchronizes $\mathcal{C}_{df,k}(S)$ with P and uses states $[\bot, q]$ for transitions that are not in P. Each state $[p_P, q]$ in N is bisimilar to state q in $\mathcal{C}_{df,k}(S)$. See Figure 5.4.

To prove that $R \sqsubseteq_{df,k} M$, we show how to obtain R from M using rules $SF(1)$, $F(1)$, $F(2)$, $F(3)$, $F(4)$, $F(5)$, and $F(6)$ defined in Section 4.5. Each rule guarantees to preserve stable failures refinement under transformation. As stable failures refinement implies deadlock freedom inclusion for each $k \in \mathbb{N}$ (Lemma 4.34), each rule also guarantees to preserve k-deadlock freedom inclusion under transformation.

Consider every reachable state $[q_P, q]$ in M. Each state $[q_P, q]$ in M is simulated by state q in *df-mp*$_k(S)$ and there exists a choice $\beta \in \varphi_k(q)$ such that state $[q_P, q]$ structurally corresponds to β. As M and $\mathcal{C}_{df,k}(S)$ are bisimilar, by construction state $[q_P, q]$ in M offers a non-deterministic alternative between all choices $\beta \in \varphi_k(q)$.

- Using rule $SF(1)$: we can remove from state $[q_P, q]$ in M all τ-branches except the one leading to the state $[q_P, q_\beta]$. See Figure 5.4 and Figure 4.9.
- Using rule $F(1)$: we can remove from state $[q_P, q]$ in M a τ-transition. See Figure 5.4 and Figure 4.9.
- Using rule $F(2)$: we can extend a τ-loop at state $[q_P, q]$ in M with an intermediate τ transition. See Figure 4.16.
- Using rule $F(3)$: we can insert a τ-loop at state $[q_P, q]$ in M. See Figure 4.16.
- Using rule $F(4)$: we can remove from state $[q_P, q]$ in M an intermediate τ transition that reaches a state which offers redundant events at another state reachable from $[q_P, q_\beta]$. See Figure 4.16.
- Using rule $F(5)$: we can remove other paths from M that are alternatives to an internal τ transition that reaches a state which enables the same events enable at state $[q_P, q_\beta]$ plus an additional τ event that leads back to state $[q_P, q]$.
- Using rule $F(6)$: we can construct a new service that offers alternatives to two copies of M.

Thus, we conclude that $\mathcal{C}_{df,k}(S)$ is a maximal k-deadlock-free controller of S. □

In the following, we prove that the canonical k-deadlock-free controller $\mathcal{C}_{df,k}(S)$ of service S is indeed a finite maximal k-deadlock-free controller of service S for each message bound $k \in \mathbb{N}$.

Theorem 5.11 ($\mathcal{C}_{df,k}(S)$ is a solution for a maximal k-deadlock-free controller of S).
For each message bound $k \in \mathbb{N}$ and each k-deadlock-freely controllable service S, the canonical k-deadlock-free service $\mathcal{C}_{df,k}(S)$ of S is a maximal k-deadlock-free controller of S, i. e., it is a solution for a maximal k-deadlock-free controller of S as defined in Definition 5.8.

Proof. For each $P \in df_k Controllers(S)$ we have $P \sqsubseteq_{df,k} \mathcal{C}_{df,k}(S)$ by construction of $\mathcal{C}_{df,k}(S)$ (Lemma 5.10). As $\mathcal{C}_{df,k}(S) \in df_k Controllers(S)$ also holds (Corollary 4.2), we conclude that $\forall P :: (P \in df_k Controllers(S)) \Leftrightarrow (P \sqsubseteq_{df,k} \mathcal{C}_{df,k}(S))$. □

As discussed in Chapter 4, the controller $\mathcal{C}_{df,k}(S)$ of S is by construction relatively large in size in comparison to an operating guideline of S. For further operations on a maximal controller, such as described by Theorem 5.6 and Theorem 5.7, it is more desirable to construct a maximal controller that is essentially small in size.

To this respect, we propose an alternative maximal k-deadlock-free controller of a given service S, that is, a compact canonical k-deadlock-free controller $\widehat{\mathcal{C}}_{df,k}(S)$ of S as defined in Section 4.3, Chapter 4. The controller $\widehat{\mathcal{C}}_{df,k}(S)$ of S is relatively small in size in comparison to the controller $\mathcal{C}_{df,k}(S)$; however, both controllers are equivalent under stable failures and under k-deadlock-freedom (cf. Lemma 4.31).

In the following theorem, we prove that the compact controller $\widehat{\mathcal{C}}_{df,k}(S)$ is also a maximal k-deadlock-free controller of service S for each message bound $k \in \mathbb{N}$.

Theorem 5.12 ($\widehat{\mathcal{C}}_{df,k}(S)$ is a solution for a maximal k-deadlock-free controller of S).
For each message bound $k \in \mathbb{N}$ and each k-deadlock-freely controllable service S, the compact canonical k-deadlock-free service $\widehat{\mathcal{C}}_{df,k}(S)$ of S is a maximal k-deadlock-free controller of S, i. e., it is a solution for a maximal k-deadlock-free controller of S as defined in Definition 5.8.

Proof. Because $\mathcal{C}_{df,k}(S)$ and $\widehat{\mathcal{C}}_{df,k}(S)$ are equivalent under stable failures (Lemma 4.31) and stable failures refinement implies deadlock freedom preorder (Lemma 4.34), it follows that $\mathcal{C}_{df,k}(S)$ and $\widehat{\mathcal{C}}_{df,k}(S)$ are also equivalent under k-deadlock freedom. Because $\mathcal{C}_{df,k}(S)$ is a solution for a maximal k-deadlock-free controller of S (Theorem 5.11), $\widehat{\mathcal{C}}_{df,k}(S)$ is also a solution for a maximal k-deadlock-free controller of S. □

Example 5.13. Consider message bound $k = 1$, service S_1 from Figure 4.1, and its k-deadlock-free operating guideline $mp_{df,k}(S_1)^{\psi^*}$ illustrated in Figure 4.2 (a). A complete canonical k-deadlock-free controller $\mathcal{C}_{df,k}(S_1)$ of S_1 is illustrated in Figure 4.2 (b) and a compact canonical k-deadlock-free controller $\widehat{\mathcal{C}}_{df,k}(S_1)$ of S_1 is illustrated in Figure 4.2 (c). Both controllers are maximal k-deadlock-free controllers of S_1 and can be constructed from the operating guideline $mp_{df,k}(S_1)^{\psi^*}$ of S_1. ◁

5.1.3. Constructing a Maximal Responsive Controller

In this section, we consider k-responsiveness as a behavioral compatibility criterion ($\mathcal{B} = rp_k$) and present construction procedures of two maximal k-responsive controllers of a given service. We prove that each constructed controller is indeed a solution for a maximal k-responsive controller of a given service for each message bound $k \in \mathbb{N}$.

To this respect, we first define a maximal k-responsive controller of a service as an instance of Definition 5.1 for $\mathcal{B} = rp_k$.

Definition 5.14 (Maximal k-responsive controller).
For each message bound $k \in \mathbb{N}$, a maximal k-responsive controller of a k-responsive controllable service S is a service $max_{rp,k}(S)$ that satisfies the following property:

$$(\forall P \in rp_k Controllers_k(S) : P \sqsubseteq_{rp,k} max_{rp,k}(S)).$$

Given a k-responsively controllable service S, we show with the following lemma that there exists an (infinite) maximal k-responsive controller of S. The proof of the lemma follows analogously from the proof of Lemma 5.9 in case of deadlock freedom.

Lemma 5.15 (Existence of a maximal k-responsive controller).
For each message bound $k \in \mathbb{N}$ and each k-responsively controllable service S : there exists a maximal responsive controller of S.

Proof. Let $P^* = sum(rp_k Controllers(S))$ be constructed from the set $rp_k Controllers(S)$ of S by applying Rule $F(6)$ (Definition 4.80). The justification of Rule $F(8)$ (Lemma 4.81) guarantees that for each $P \in rp_k Controllers(S)$ holds: $P^* \sqsubseteq_{RF} P$. That is, each $P \in rp_k Controllers(S)$ refines P^* under responsive failures. As responsive failures refinement implies responsiveness Inclusion (Lemma 4.56), it follows that P is smaller than or equal to P^* in the responsive preorder, that is, $P \sqsubseteq_{rp,k} P^*$ holds.

Assume P^* is not a responsive controller of S, i. e., $P^* \notin rp_k Controllers(S)$. By definition, the composition $S \oplus P^*$ is not k-responsive. This means, there is a state q that is either a deadlock state or a non-responsive state in either $Beh_{S \oplus P^*}(S)$ or $Beh_{S \oplus P^*}(P^*)$. Certainly, this state q does not involve the initial state of P^* as the initial state offers non-deterministic τ choices of all controllers in $rp_k Controllers(S)$. As S is k-responsively controllable; by definition $rp_k Controllers(S) \neq \emptyset$. By construction, P^* offers a non-deterministic τ choice between all services in $rp_k Controllers(S)$. As $S \oplus P^*$ is not k-responsive by assumption, it follows that there must be a service P in the set $rp_k Controllers_k(S)$ such that neither $Beh_{S \oplus P^*}(S)$ nor $Beh_{S \oplus P^*}(P^*)$ contains either a deadlock state or a divergent state. This means, $\neg rp(S \oplus P)$ must hold, which contradicts to the assumption $P \in rp_k Controllers(S)$.

Thus, $P^* = sum(rp_k Controllers(S))$ is indeed an (infinite) maximal responsive controller of S. □

Next, we recall a canonical k-responsive controller $\mathcal{C}_{rp,k}(S)$ of service S discussed in Section 4.1, of a given service S. The controller $\mathcal{C}_{rp,k}(S)$ is constructed from a finite representation of all k-responsive controllers of S (known as a responsive operating guideline of S) by replacing each labeled state q with a fragment of of nondeterministic internal alternative between all valid responsive choices that is described at state q. Note, that this construction procedure is similar to the construction of $\mathcal{C}_{df,k}(S)$ in case of deadlock freedom.

In Theorem 5.17, we prove that the canonical k-responsive controller $\mathcal{C}_{rp,k}(S)$ is indeed a maximal k-responsive controller of service S. As $\mathcal{C}_{rp,k}(S)$ is a k-responsive controller of S (Corollary 4.7), we first prove in Lemma 5.16 that the controller $\mathcal{C}_{rp,k}(S)$ is larger than or equal to every k-responsive controller of S.

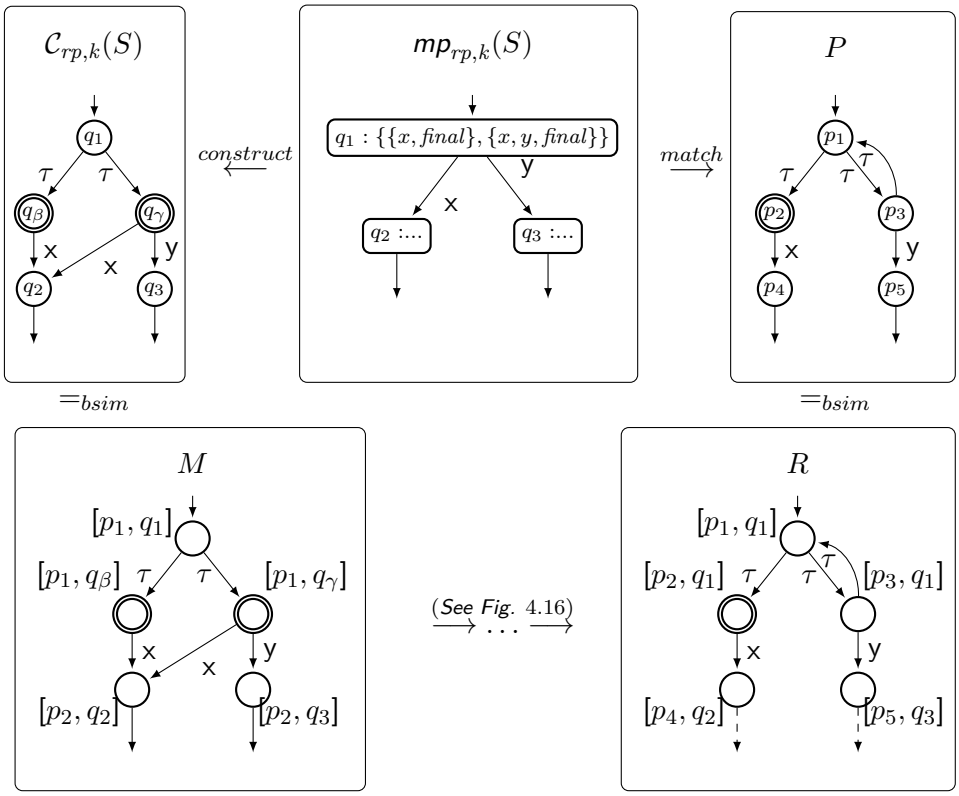


Figure 5.5.: A service P that matches with $mp_{rp,k}(S)$ can be obtained from $\mathcal{C}_{rp,k}(S)$.

Lemma 5.16 ($\mathcal{C}_{rp,k}(S)$ is a maximal k-responsive controller of S).
For each message bound $k \in \mathbb{N}$ and each k-responsively controllable service S:

$$(\forall P :: P \in rp_k Controllers_k(S)) \;\Rightarrow\; (P \;\sqsubseteq_{rp,k}\; \mathcal{C}_{rp,k}(S))).$$

Proof. Let S and $P = [Q_P, q_{0P}, I, O, \rightarrow_P, \Omega_P]$ be two k-responsively controllable services with compatible interfaces. Let $mp_{rp,k}(S)^{\psi^*} = [mp_{rp,k}(S), \psi^*] = [Q, q_0, I, O, \rightarrow, \Omega, \psi^*]$ be

a k-responsive operating guideline S, and $\mathcal{C}_{rp,k}(S)$ be constructed from $mp_{rp,k}(S)^{\psi^*}$ as described in Definition 4.6.

Let $R = [Q_R, q_{0R}, I, O, \rightarrow_R, \Omega_R]$ be a service with

- $Q_R = q_P \times Q$;
- $q_{0R} = [q_{0P}, q]$;
- $\Omega_R = \Omega_P \times Q$;
- $\rightarrow_R = \{[q_P, q] \xrightarrow{m}_R [q'_P, q'] \mid m \neq \tau \ \wedge \ q_P \xrightarrow{m}_P q'_P \ \wedge \ q \xrightarrow{m} q'\} \cup$
  $\{[q_P, q] \xrightarrow{\tau}_R [q'_P, q] \mid q_P \xrightarrow{\tau}_P q'_P\}$;

Service R synchronizes P with $mp_{rp,k}(S)$ but ignores transitions that are not shared. Each state $[p_P, q]$ in N is bisimilar to state q_P in P as $mp_{rp,k}(S)$ structurally simulates P and $mp_{rp,k}(S)$ is deterministic. See Figure 5.5.

Let $M = [Q_M, q_{0M}, I, O, \rightarrow_M, \Omega_M]$ be a service with

- $Q_M = (q_P \cup \{\bot\}) \times (Q \cup \{q_\gamma \mid q \in Q \ \wedge \ \gamma \in \psi^*_k(q)\})$;
- $q_{0M} = [q_{0P}, q]$;
- $\Omega_M = (q_P \cup \{\bot\}) \times \{Q_\gamma \mid \gamma \in \psi^*_k(q)\}$;
- $\rightarrow_M = \{[q_P, q] \xrightarrow{\tau}_M [q_P, q_\gamma] \mid q \in Q \wedge \gamma \in \psi^*_k(q)\} \cup$
  $\{[q_P, q_\gamma] \xrightarrow{m}_M [q'_P, q'] \mid q \xrightarrow{m} q' \ \wedge \ \gamma \in \psi^*_k(q) \wedge$
  $(q_P \xrightarrow{m}_P q'_P \vee (q'_P = \bot \ \wedge \neg(\exists q''_P :: q_P \xrightarrow{m}_P q''_P)))\}$

Service M synchronizes $\mathcal{C}_{rp,k}(S)$ with P and uses states $[\bot, q]$ for transitions that are not in P. Each state $[p_P, q]$ in M is bisimilar to state q in $\mathcal{C}_{rp,k}(S)$. See Figure 5.5.

To prove that $R \sqsubseteq_{rp,k} M$, we show how to obtain R from M using rules $SF(1)$, $F(1)$, $F(2)$, $F(3)$, $F(3)$, $F(4)$, $F(5)$, and $F(6)$; Each rule is defined in Section 4.5 and guarantees to preserve responsive failures refinement under transformation. As responsive failures refinement implies k-responsive inclusion for each $k \in \mathbb{N}$ (Lemma 4.56), each rule also guarantees to preserve k-responsiveness inclusion under transformation.

Consider every reachable state $[q_P, q]$ in R. Each state $[q_P, q]$ in R structurally simulates state q in $mp_{rp,k}(S)$ and there exists $\gamma \in \psi^*_k(q)$ such that either $\gamma \in \psi^*_k(q)$ with either $\gamma = act^*(\tau(q_P))$. As M and $\mathcal{C}_{rp,k}(S)$ are bisimilar, by construction M offers a non-deterministic alternative between all choices $\gamma \in \psi^*(q)$.

- Using rule $F(3)$: we can insert a τ-loop transition at state $[q_P, q]$ in M. See Figure 4.16.
- Using rule $F(4)$: we can remove from state $[q_P, q]$ in M an intermediate τ transition that reaches a state which enables a superset of all events that are enabled at state $[q_P, q_\beta]$ reachable from $[q_P, q]$ by a τ transition. See Figure 4.13 and Figure 4.16.
- Using rule $F(2)$: we can extend a τ-loop at state $[q_P, q]$ in M with an intermediate τ transition. See Figure 5.5 and Figure 4.16.
- Using rule $RF(1)$: we can remove from state $[q_P, q]$ in M all τ-branches except the one leading to the state $[q_P, q_\beta]$. See Figure 4.13 and Figure 4.16.
- Using rule $F(1)$: we can remove from state $[q_P, q]$ in M a τ-transition. See Figure 4.13.

– Using rule $F(6)$: we can construct a new service that offers alternatives to two copies of M.

Thus, we conclude that $\mathcal{C}_{rp,k}(S)$ is a maximal k-responsive controller of S. □

In the following theorem, we prove that the constructed service $\mathcal{C}_{rp,k}(S)$ of service S is indeed a finite maximal k-responsive controller of service S for each message bound $k \in \mathbb{N}$.

Theorem 5.17 ($\mathcal{C}_{rp,k}(S)$ is a solution for a maximal k-responsive controller of S).
For each message bound $k \in \mathbb{N}$ and each k-responsively controllable service S, the service $\mathcal{C}_{rp,k}(S)$ is a maximal k-responsive controller of S, i. e., it is a solution for $\mathcal{C}_{rp,k}(S)$ in Definition 5.14.

Proof. For each $P \in rp_k Controllers(S))$ we have $P \sqsubseteq_{rp,k} \mathcal{C}_{rp,k}(S)$ by construction of $\mathcal{C}_{rp,k}(S)$ (Lemma 5.16). As $\mathcal{C}_{rp,k}(S) \in rp_k Controllers(S)$ also holds (Corollary 4.7), we conclude that $\forall P :: (P \in rp_k Controllers(S)) \Leftrightarrow (P \sqsubseteq_{rp,k} \mathcal{C}_{rp,k}(S)))$. □

As discussed in Chapter 4, the controller $\mathcal{C}_{rp,k}(S)$ of S is by definition relatively large in size by construction in comparison to a responsive operating guideline of S. Similar to deadlock freedom, is desirable to construct a maximal controller that is essentially small in its size for further operations on a maximal controller, such as described by Theorem 5.6 and Theorem 5.7.

To this respect, we propose an alternative construction procedure of a compact canonical k-responsive controller $\widehat{\mathcal{C}}_{rp,k}(S)$ as defined in Section 4.8, Chapter 4. The controller $\widehat{\mathcal{C}}_{rp,k}(S)$ of S is relatively small in size in comparison to the controller $\mathcal{C}_{rp,k}(S)$; however, both controllers are equivalent under responsive failures and under k-responsiveness.

In the following theorem, we prove that the compact canonical k-responsive controller $\widehat{\mathcal{C}}_{rp,k}(S)$ is also a maximal k-responsive controller of service S for each message bound $k \in \mathbb{N}$.

Theorem 5.18 ($\widehat{\mathcal{C}}_{rp,k}(S)$ is a solution for a maximal k-responsive controller of S).
For each message bound $k \in \mathbb{N}$ and each k-responsively controllable service S, the compact canonical k-responsive service $\widehat{\mathcal{C}}_{rp,k}(S)$ of S is a maximal k-responsive controller of S, i. e., it is a solution for a maximal k-responsive controller of S defined in Definition 5.14.

Proof. Because $\mathcal{C}_{rp,k}(S)$ and $\widehat{\mathcal{C}}_{rp,k}(S)$ are equivalent under responsive failures (Corollary 4.53) and responsive failures refinement implies responsiveness preorder (Lemma 4.56), it follows that $\mathcal{C}_{rp,k}(S)$ and $\widehat{\mathcal{C}}_{rp,k}(S)$ are also equivalent under k-responsiveness. Because $\mathcal{C}_{rp,k}(S)$ is a solution for a maximal k-responsive controller of S (Theorem 5.17), then $\widehat{\mathcal{C}}_{rp,k}(S)$ is also a solution for a maximal k-responsive controller of S. □

Example 5.19. Consider message bound $k = 1$, service S_1 from Figure 4.1, and its k-responsive operating guideline $mp_{rp,k}(S_1)^{\psi^*}$ illustrated in Figure 4.3 (a). A complete canonical k-responsive controller $\mathcal{C}_{rp,k}(S_1)$ of S_1 is illustrated in Figure 4.3 (b) and a compact canonical k-responsive controller $\widehat{\mathcal{C}}_{rp,k}(S_1)$ of S_1 is illustrated in Figure 4.3 (c). Both controllers are maximal k-responsive controllers of S_1 and can be constructed from the operating guideline $mp_{rp,k}(S_1)^{\psi^*}$ of S_1. ◁

5.2. Representing Substitutes for Services

In the previous section, we presented a maximal $\mathcal{B}$-controller of a given service and its distinguished properties. We show that a canonical $\mathcal{B}$-controller of a given service is a solution for its maximal $\mathcal{B}$-controller for each behavioral compatibility criterion $\mathcal{B} = \{df_k, rp_k\}$ and each message bound $k \in \mathbb{N}$.

In this section, we illustrate how to employ the results from the previous section to represent the set of all services that can substitute the given service under $\mathcal{B}$ inclusion. In Section 5.2.1, we present an operating guideline of a canonical controller of service S as a finite representation of all services that can substitute S under $\mathcal{B}$ inclusion. In Section 5.2.2, we present a canonical substitution service of S as a service that can canonically represent all services that can substitute S under $\mathcal{B}$ inclusion. We illustrate the experimental results of constructing an operating guideline of a canonical controller and a canonical substitute in case of responsiveness ($\mathcal{B} = rp$) in Section 5.2.3.

5.2.1. Using Operating Guidelines and Canonical Controller

One of the results from the previous section (cf. Theorem 5.7) illustrates that the set of all substitution services of a given service S under $\mathcal{B}$ inclusion can be characterized in terms of the set of all $\mathcal{B}$-controllers of a maximal $\mathcal{B}$-controller $\mathcal{B}max(S)$.

For each behavioral compatibility criterion $\mathcal{B} = \{df_k, rp_k\}$, we can employ a canonical $\mathcal{B}$-controller of the given service as a solution for its maximal $\mathcal{B}$-controller (in case $\mathcal{B} = df_k$ cf. Lemma 5.10, Theorem 5.12; and in case $\mathcal{B} = rp_k$ cf. Lemma 5.16 and Theorem 5.18).

In this section, we show how to realize Theorem 5.7 by combining the construction of a canonical $\mathcal{B}$-controller with the operating guidelines technique [Massuthe and Schmidt, 2005, Lohmann et al., 2007a, Massuthe, 2009, Lohmann, 2010] introduced in Section 3.3 to construct a finite representation of the set of services that can substitute a given service under $\mathcal{B}$-inclusion. Note, that the realization in the two following subsections share similar techniques for representing all substitutes for a given service, though for different behavioral compatibility criterion.

Deadlock-free Operating Guideline and Canonical Deadlock-free Controller

In this section, we realize Theorem 5.7 for $\mathcal{B} = df_k$ by combining the construction of a canonical k-deadlock-free controller with the operating guidelines technique to construct a finite representation of the set of services that can substitute a given service under k-deadlock-free inclusion.

For a given service S, a deadlock-free operating guideline of S is a finite representation of all k-deadlock-free controllers of S for each message channel $k \in \mathbb{N}$. Nevertheless, two services that are equivalent under k-deadlock freedom may have different k-deadlock-free operating guidelines, though each represents the same set of k-deadlock-free-controllers. As a result, the construction procedure of canonical k-deadlock-free controllers of the two k-deadlock-freely equivalent services may yield two different canonical k-deadlock-free-controllers.

For each message bound $k \in \mathbb{N}$, we first show that two services are equivalent under k-deadlock freedom whenever their canonical k-deadlock-free controllers are bisimilar.

Lemma 5.20 (Complete canonical k-deadlock-free controllers are bisimilar). For each message bound $k \in \mathbb{N}$ and each two k-deadlock-freely controllable services S and T with equivalent interface:

$$S =_{df,k} T \quad \Leftrightarrow \quad \mathcal{C}_{df,k}(S) =_{bsim} \mathcal{C}_{df,k}(T).$$

Proof. We prove this theorem in two directions:

$\Rightarrow$: Suppose $S =_{df,k} T$. By definition of $=_{df,k}$, the deadlock-free operating guidelines of S and T have the same sets of strong compliance matching services. Using Corollary 3.53 this means that the two underlying automata of operating guidelines strongly simulate each other, and their sets of deadlock-free choices of the simulated states are equivalent. As an underlying automaton of an operating guideline is deterministic, we conclude that $\mathcal{C}_{df,k}(S) =_{bsim} \mathcal{C}_{df,k}(T)$ by construction (Definition 4.1).

$\Leftarrow$: Suppose $\mathcal{C}_{df,k}(S) =_{bsim} \mathcal{C}_{df,k}(T)$. As bisimulation implies deadlock freedom equivalence, $\mathcal{C}_{df,k}(S) =_{df,k} \mathcal{C}_{df,k}(T)$ follows. This implies, both $\mathcal{C}_{df,k}(T) \sqsubseteq_{df,k} \mathcal{C}_{df,k}(S)$ and $\mathcal{C}_{df,k}(S) \sqsubseteq_{df,k} \mathcal{C}_{df,k}(T)$ hold. As $\mathcal{C}_{df,k}(T)$ is a solution for $max_{df,k}(S)$ and $\mathcal{C}_{df,k}(T)$ is a solution for $max_{df,k}(T)$ (Theorem 5.11), by applying Theorem 5.3 for $B = df$ twice, we respectively obtain $S \sqsubseteq_{df,k} T$ and $T \sqsubseteq_{df,k} S$. Thus, $S =_{df,k} T$ holds.

Thus, the theorem holds. □

Lemma 5.20 illustrates that the canonical k-deadlock-free controllers can be used as a canonical representative of all k-deadlock-free controllers of the k-deadlock freedom equivalence class of service S. A similar phenomenon holds for the construction procedure of compact k-deadlock-free controllers.

Corollary 5.21 (Compact canonical k-deadlock-free controllers are bisimilar). For each message bound $k \in \mathbb{N}$ and each two k-deadlock-freely controllable services S and T with equivalent interface:

$$S =_{df,k} T \quad \Leftrightarrow \quad \widehat{\mathcal{C}}_{df,k}(S) =_{bsim} \widehat{\mathcal{C}}_{df,k}(T).$$

Proof. The proof of this corollary follows from Lemma 5.20. □

In the two followings corollaries, we illustrate some useful applications of a canonical k-deadlock-free controller of a given service that immediately follows from the properties of maximal k-deadlock-free controller discussed in Section 5.1.1.

Corollary 5.22 (Deciding substitutability under k-deadlock freedom). For each message bound $k \in \mathbb{N}$ and each k-deadlock-freely controllable service S :

$$T \sqsubseteq_{df,k} S \Leftrightarrow df_k(T \oplus \mathcal{C}_{df,k}(S)) \quad \text{and} \quad T \sqsubseteq_{df,k} S \Leftrightarrow df_k(T \oplus \widehat{\mathcal{C}}_{df,k}(S)).$$

Proof. Follows from Theorem 5.6 (for $\mathcal{B} = df_k$), Theorem 5.11, and Lemma 5.12. □

Corollary 5.22 realizes Theorem 5.6 for deadlock freedom ($B = df_k$). We first construct either a complete canonical k-deadlock free controller $\mathcal{C}_{df,k}(S)$ or a compact canonical k-deadlock free controller $\widehat{\mathcal{C}}_{df,k}(S)$ from a given service S. Then we employ the constructed controller to decide substitutability of service T under k-deadlock freedom preorder. The decision procedure is done by checking whether the composition of T and the constructed controller satisfies the deadlock freedom property.

As the decision procedure involves checking the property of service composition, obviously the compact canonical k-deadlock free controller it is more desirable as $\widehat{\mathcal{C}}_{df,k}(S)$ is relatively smaller in size than the complete canonical k-deadlock free controller $\mathcal{C}_{df,k}(S)$ of service S.

Corollary 5.23 (Characterization of substitutes under k-deadlock freedom). For each message bound $k \in \mathbb{N}$ and each k-deadlock-freely controllable service S :

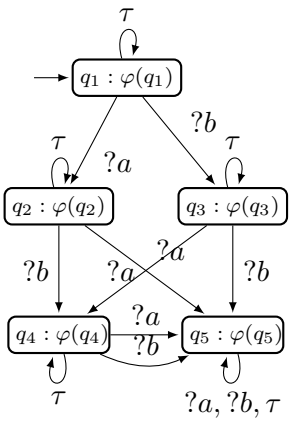
$$\begin{aligned} df_k Inclusion(S) &= df_k Controllers(\mathcal{C}_{df,k}(S)) = Comply_\beta(mp_{df,k}(\mathcal{C}_{df,k}(S))^\varphi) \\ &= df_k Controllers(\widehat{\mathcal{C}}_{df,k}(S)) = Comply_\beta(mp_{df,k}(\widehat{\mathcal{C}}_{df,k}(S))^\varphi) \end{aligned}$$

where $mp_{df,k}(\mathcal{C}_{df,k}(S))^\varphi$ and $mp_{df,k}(\widehat{\mathcal{C}}_{df,k}(S))^\varphi$ are k-deadlock-free operating guidelines of $\mathcal{C}_{df,k}(S)$ and of $\widehat{\mathcal{C}}_{df,k}(S)$ respectively.

Proof. Follows from Theorem 5.7 (for $\mathcal{B} = df_k$), Theorem 5.11, and Theorem 5.12, and Lemma 3.51. □

The value of Corollary 5.23 is the characterization of all services that can substitute a given service under k-deadlock freedom inclusion. This means we can construct a finite representation of all substitution services of service S under k-deadlock freedom inclusion as an k-deadlock-free operating guideline of a canonical k-deadlock-free controllers of service S.

Similar to the procedure described in Corollary 5.22, a compact canonical k-deadlock free controller $\widehat{\mathcal{C}}_{df,k}(S)$ is more desirable for the procedure described in Corollary 5.23 as $\widehat{\mathcal{C}}_{df,k}(S)$ is relatively smaller in size than the complete canonical k-deadlock free controller $\mathcal{C}_{df,k}(S)$ of service S.



$$
\begin{aligned}
\varphi(q_1) &= \{\ \{?a\},\{?b\},\{\tau\},\{?a,?b\},\{?a,\mathit{final}\},\\
&\quad \{?a,\mathit{final}\},\{?b,\tau\},\{?b,\mathit{final}\},\{\tau,\mathit{final}\},\\
&\quad \{?a,?b,\tau\},\{?a,\tau,\mathit{final}\},\{?a,?b,\mathit{final}\},\\
&\quad \{?b,\tau,\mathit{final}\},\{?a,?b,\tau,\mathit{final}\}\ \},\\
\varphi(q_2) &= \{\ \{?b\},\{\tau\},\{?b,\tau\},\{?b,\mathit{final}\},\{?b,?a\},\\
&\quad \{\tau,\mathit{final}\},\{\tau,?a\},\{?b,\tau,?a\},\{?b,\mathit{final},?a\},\\
&\quad \{?b,\tau,\mathit{final}\},\{\tau,\mathit{final},?a\},\{?b,\tau,\mathit{final},?a\}\ \}\\
\varphi(q_3) &= \{\ \{?a\},\{\tau\},\{?a,\tau\},\{?a,\mathit{final}\},\{?a,?b\},\\
&\quad \{\tau,\mathit{final}\},\{\tau,?b\},\{?a,\tau,?a\},\{?a,\mathit{final},?b\},\\
&\quad \{?b,\tau,\mathit{final}\},\{\tau,\mathit{final},?b\},\{?a,\tau,\mathit{final},?b\}\ \}\\
\varphi(q_4) &= \{\ \{\mathit{final}\},\{\tau\},\{\mathit{final},?a\},\{\mathit{final},?b\},\\
&\quad \{\mathit{final},\tau\},\{\tau,?a\},\{\tau,?b\},\\
&\quad \{\mathit{final},\tau,?a\},\{\mathit{final},\tau,?b\},\{\mathit{final},?a,?b\},\\
&\quad \{\tau,?a,?b\},\{\mathit{final},\tau,?a,?b\}\ \}
\end{aligned}
$$

$$
\begin{aligned}
\varphi(q_5) = \{\ &\{\ \},\{?a\},\{?b\},\{\mathit{final}\},\{\tau\},\\
&\{?a,?b\},\{?a,\mathit{final}\},\{?a,\tau\},\{?b,\tau\},\{?b,\mathit{final}\},\{\mathit{final},\tau\},\\
&\{?a,?b,\mathit{final}\},\{?a,?b,\tau\},\{?a,\mathit{final},\tau\},\{?b,\mathit{final},\tau\},\\
&\{?a,?b,\mathit{final},\tau\}\ \}
\end{aligned}
$$

Figure 5.6.: A k-deadlock-free operating guidelines $mp_{df,k}(\mathcal{C}_{df,k}(S_1))^{\varphi}$ of a canonical k-deadlock-free controller $\mathcal{C}_{df,k}(S_1)$ (Figure 4.2) of S_1 (Figure 4.1) for $k = 1$.

Example 5.24. Figure 5.6 illustrates an operating guideline $mp_{df,k}(\mathcal{C}_{df,k}(S_1))^{\varphi}$ of a canonical k-deadlock-free controller $\mathcal{C}_{df,k}(S_1)$ (from Figure 4.2) of service S_1 from Figure 4.1 for message bound $k = 1$. The underlying most permissive k-deadlock-free controller $mp_{df,k}(\mathcal{C}_{df,k}(S_1))$ has an interface, $I = \{a, b\}$ and $O = \{\ \}$, that is, $mp_{df,k}(\mathcal{C}_{df,k}(S_1))$ and S_1 has the same interface whereas $mp_{df,k}(\mathcal{C}_{df,k}(S_1))$ and $\mathcal{C}_{df,k}(S_1)$ has compatible interface. For readability purpose, the interface is omitted from the figure.

Consider services S_2, S_3, and S_4 from Figure 4.1. We see that each of them strongly complies with $mp_{df,k}(\mathcal{C}_{df,k}(S_1))^{\varphi}$. This means, each service is a k-deadlock-free controller of the controller $\mathcal{C}_{df,k}(S_1)$. With Corollary 5.23, we conclude that each of them can substitute service S_1 under k-deadlock freedom inclusion.

Note that an operating guideline $mp_{df,k}(\widehat{\mathcal{C}}_{df,k}(S_1))^{\varphi}$ of compact canonical k-deadlock-free controller $\widehat{\mathcal{C}}_{df,k}(S_1)$ (from Figure 4.2) of service S_1 is equivalent to the operating guideline shown in Figure 5.6. Only if the construction of $mp_{df,k}(\widehat{\mathcal{C}}_{df,k}(S_1))^{\varphi}$ is more efficient as $\widehat{\mathcal{C}}_{df,k}(S_1)$ is smaller in size than $\mathcal{C}_{df,k}(S_1)$. ◁

The finite representation of all substitutes for a given service under k-deadlock freedom has several useful applications. We discuss further the respective applications of a canonical deadlock-free controller in Chapter 7.

Responsive Operating Guideline and Canonical Responsive Controller

In this section, we realize Theorem 5.7 for $\mathcal{B} = rp_k$ by combining the construction of a canonical k-responsive controller with the operating guidelines technique to finitely

represent the services that can substitute a given service under k-responsive inclusion. For a given service S, a responsive operating guideline of S is a finite representation of all its k-responsive controllers for each message channel $k \in \mathbb{N}$. Similarly to deadlock freedom, two k-responsively equivalent services may have different k-responsive operating guidelines, though each represents the same set of k-responsive-controllers. As a result, the construction procedure of a canonical k-responsive controller of the two k-responsively equivalent services may yield two different canonical k-responsive controllers.

In the followings, we show that two services are equivalent under k-responsiveness whenever their canonical k-responsive controllers are bisimilar.

Lemma 5.25 (Canonical k-responsive controllers are bisimilar).
For each message bound $k \in \mathbb{N}$ and each two k-responsively controllable services S and T:

$$S =_{rp,k} T \quad \Leftrightarrow \quad \mathcal{C}_{rp,k}(S) =_{bsim} \mathcal{C}_{rp,k}(T).$$

Proof. We prove this theorem in two directions:

$\Rightarrow$: Suppose $S =_{rp,k} T$. By definition of $=_{rp,k}$, a responsive operating guideline of S and T have the same sets of structural compliance matching services. Using Corollary 3.71 this means that the two underlying automaton of operating guidelines strongly simulate each other, and their sets of responsive choices of the simulated states are equivalent. As an underlying of an operating guideline is deterministic, from Definition 4.6, we conclude that $\mathcal{C}_{rp,k}(S) =_{bsim} \mathcal{C}_{rp,k}(T)$.

$\Leftarrow$: Suppose $\mathcal{C}_{rp,k}(S) =_{bsim} \mathcal{C}_{rp,k}(T)$. As bisimulation implies responsive equivalence, then $\mathcal{C}_{rp,k}(S) =_{rp,k} \mathcal{C}_{rp,k}(T)$ follows. It implies that both $\mathcal{C}_{rp,k}(T) \sqsubseteq_{rp,k} \mathcal{C}_{rp,k}(S)$ and $\mathcal{C}_{rp,k}(S) \sqsubseteq_{rp,k} \mathcal{C}_{rp,k}(T)$ holds. As $\mathcal{C}_{rp,k}(S)$ is a solution for $max_{rp,k}(S)$ and $\mathcal{C}_{rp,k}(T)$ is a solution for $max_{rp,k}(T)$ (Theorem 5.17), by applying Theorem 5.3 for $B = rp$ twice, we respectively obtain $S \sqsubseteq_{rp,k} T$ and $T \sqsubseteq_{rp,k} S$. Thus, $S =_{rp,k} T$ holds.

Thus, the theorem holds. □

Lemma 5.25 illustrates that a canonical k-responsive controller can be used as a canonical representative of all k-responsive controllers of the k-responsiveness equivalence class of service S. A similar phenomenon holds for the construction procedure of compact k-responsive controllers.

Corollary 5.26 (Compact canonical k-responsive controllers are bisimilar).
For each message bound $k \in \mathbb{N}$ and each two k-responsively controllable services S and T with equivalent interface:

$$S =_{rp,k} T \quad \Leftrightarrow \quad \widehat{\mathcal{C}}_{rp,k}(S) =_{bsim} \widehat{\mathcal{C}}_{rp,k}(T).$$

Proof. The proof of this corollary follows from Lemma 5.25. □

In the two following corollaries, we illustrate some useful applications of a canonical k-responsive controller of a given service that immediately follows from the properties of maximal k-responsive controller discussed in Section 5.1.1.

Corollary 5.27 (Deciding substitutability under k-responsiveness).
For each message bound $k \in \mathbb{N}$ and each k-responsively controllable service S :

$$T \sqsubseteq_{rp,k} S \Leftrightarrow rp_k(T \oplus \mathcal{C}_{rp,k}(S)) \quad \text{and} \quad T \sqsubseteq_{rp,k} S \Leftrightarrow rp_k(T \oplus \widehat{\mathcal{C}}_{rp,k}(S)).$$

Proof. Follows from Theorem 5.6 (for $\mathcal{B} = rp_k$), Theorem 5.17, and Theorem 5.18. □

Corollary 5.27 realizes Theorem 5.6 for responsiveness ($B = df_k$). We first construct either a complete canonical k-responsive controller $\mathcal{C}_{rp,k}(S)$ or a compact canonical k-responsive controller $\widehat{\mathcal{C}}_{rp,k}(S)$ from a given service S. Then we employ the constructed controller to decide substitutability of service T under k-responsiveness preorder. The decision procedure is done by checking whether the composition of T and the constructed controller satisfies the responsiveness property.

As the decision procedure involves checking the property of service composition, obviously the compact canonical k-responsive controller $\widehat{\mathcal{C}}_{rp,k}(S)$ is more desirable as $\widehat{\mathcal{C}}_{rp,k}(S)$ is relatively smaller in size than the complete canonical k-responsive controller $\mathcal{C}_{rp,k}(S)$ of service S.

Corollary 5.28 (Characterization of substitutes under k-responsiveness).
For each message bound $k \in \mathbb{N}$ and each k-responsively controllable service S :

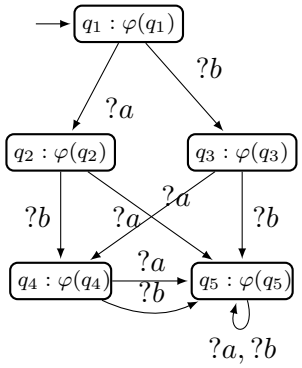
$$\begin{aligned} rp_k Inclusion(S) &= df_k Controllers(\mathcal{C}_{rp,k}(S)) = Comply_\beta(mp_{rp,k}(\mathcal{C}_{rp,k}(S))^\varphi) \\ &= df_k Controllers(\widehat{\mathcal{C}}_{rp,k}(S)) = Comply_\beta(mp_{rp,k}(\widehat{\mathcal{C}}_{rp,k}(S))^\varphi) \end{aligned}$$

where $mp_{rp,k}(\mathcal{C}_{rp,k}(S))^{\psi^*}$ and $mp_{rp,k}(\widehat{\mathcal{C}}_{rp,k}(S))^{\psi^*}$ are k-responsive operating guidelines of $\mathcal{C}_{rp,k}(S)$ and of $\widehat{\mathcal{C}}_{rp,k}(S)$ respectively.

Proof. Follows from Theorem 5.7 (for $\mathcal{B} = rp_k$), Theorem 5.17, and Theorem 5.18, and Lemma 3.69. □

Corollary 5.28 has a value for characterizing of all services that can substitute a given service under k-responsiveness inclusion. This means we can construct a finite representation of all substitution services of service S under k-responsiveness inclusion as a k-responsive operating guideline of a canonical k-responsive controller of service S.

Similar to the procedure described in Corollary 5.27, a compact canonical k-responsive controller $\widehat{\mathcal{C}}_{rp,k}(S)$ is more desirable for the procedure described in Corollary 5.28 as it is relatively smaller in size than the complete canonical k-responsive controller $\mathcal{C}_{rp,k}(S)$ of service S.



$$
\begin{aligned}
\varphi(q_1) &= \{\ \{?a\},\{?b\},\{?a,?b\},\{?a,\mathit{final}\},\\
&\quad\ \{?b,\mathit{final}\},\{?a,?b,\mathit{final}\}\ \},\\
\varphi(q_2) &= \{\ \{?b\},\{?b,\mathit{final}\},\{?b,?a\},\{?b,\mathit{final},?a\}\ \}\\
\varphi(q_3) &= \{\ \{?a\},\{?a,\mathit{final}\},\{?a,?b\},\{?a,\mathit{final},?b\}\ \}\\
\varphi(q_4) &= \{\ \{\mathit{final}\},\{\mathit{final},?a\},\{\mathit{final},?b\},\\
&\quad\ \{\mathit{final},?a,?b\}\ \}\\
\varphi(q_5) &= \{\ \{\ \},\{?a\},\{?b\},\{\mathit{final}\},\\
&\quad\ \{?a,?b\},\{?a,\mathit{final}\},\{?b,\mathit{final}\},\\
&\quad\ \{?a,?b,\mathit{final}\}\ \}
\end{aligned}
$$

Figure 5.7.: A k-responsive operating guidelines $mp_{rp,k}(\mathcal{C}_{rp,k}(S_1))^{\psi^*}$ of a canonical k-responsive controller $\mathcal{C}_{rp,k}(S_1)$ (Figure 4.3) of S_1 (Figure 4.1) for $k = 1$.

Example 5.29. Figure 5.7 illustrates an operating guideline $mp_{rp,k}(\mathcal{C}_{rp,k}(S_1))^{\psi^*}$ of a canonical k-responsive controller $\mathcal{C}_{rp,k}(S_1)$ (from Figure 4.3) of service S_1 from Figure 4.1 for message bound $k = 1$. The underlying most permissive k-responsive controller $mp_{rp,k}(\mathcal{C}_{rp,k}(S_1))$ has an interface, $I = \{a, b\}$ and $O = \{\ \}$, this means, $mp_{rp,k}(\mathcal{C}_{rp,k}(S_1))$ and S_1 has the same interface whereas $mp_{rp,k}(\mathcal{C}_{rp,k}(S_1))$ and $\mathcal{C}_{rp,k}(S_1)$ has compatible interface. For readability purpose, the interface is omitted from the figure.

Consider services S_2, S_3, and S_4 from Figure 4.1.

We see that only services S_2 and S_3 structurally comply with $mp_{rp,k}(\mathcal{C}_{rp,k}(S_1))^{\psi^*}$, this implies, services S_2 and S_3 are both k-responsive controllers of the controller $\mathcal{C}_{rp,k}(S_1)$. With Corollary 5.23, we can conclude that each of them can substitute service S_1 under k-responsive inclusion.

Nevertheless, service S_4 structurally does not comply with $mp_{rp,k}(\mathcal{C}_{rp,k}(S_1))^{\psi^*}$. This is due to state d_3 of S_4 that cannot provide an event that is described by a choice of $\varphi(q_1)$. This means, service S_4 is not k-responsive controllers of the controller $\mathcal{C}_{rp,k}(S_1)$. With Corollary 5.28, we conclude that service S_4 cannot substitute service S_1 under k-responsive inclusion.

Note, that the operating guideline $mp_{rp,k}(\widehat{\mathcal{C}}_{rp,k}(S_1))^{\psi^*}$ of the compact canonical k-responsive controller $\widehat{\mathcal{C}}_{rp,k}(S_1)$ (from Figure 4.3) of service S_1 is equivalent to the operating guideline shown in Figure 5.6. Nevertheless, the construction of $mp_{rp,k}(\widehat{\mathcal{C}}_{rp,k}(S_1))^{\psi^*}$ is more efficient as $\widehat{\mathcal{C}}_{rp,k}(S_1)$ is smaller in size than $\mathcal{C}_{rp,k}(S_1)$. ◁

The finite representation of all substitution services under k-responsiveness has several useful applications. We discuss the respective applications of a canonical responsive controller in further Chapter 7.

5.2.2. Using Canonical Substitute for a Service

Results from the previous section (cf. Theorem 5.4) show that applying the construction of a maximal $\mathcal{B}$-controller twice is a closure operation on a given service.

With the construction procedure of a maximal $\mathcal{B}$-controller, we have constructed such a service for deadlock freedom ($\mathcal{B} = df_k$, cf. Definition 4.1 and Lemma 5.10) and responsiveness ($\mathcal{B} = rp_k\}$ cf. Definition 4.6 and Lemma 5.16).

In this section, we show how to realize Theorem 5.4 using the construction procedure of a canonical $\mathcal{B}$-controller. We denote a service that is an output of this closure operation on a given service S by a *canonical $\mathcal{B}$ substitute* for service S, for each behavioral compatibility criterion $\mathcal{B} \in \{df_k, rp_k\}$. We also illustrate that a *canonical $\mathcal{B}$ substitute* for S represents the set of all substitutes for S under $\mathcal{B}$ inclusion. Note, that the two following subsections share similar techniques, though for different behavioral compatibility criterion.

Canonical Deadlock-free Substitute for a Service

In this section, we denote a *complete canonical k-deadlock-free substitute* for service S by $\mathcal{C}^2_{df,k}(S)$ and a *compact canonical k-deadlock-free substitute* for service S by $\widehat{\mathcal{C}}^2_{df,k}(S)$. Each service can be constructed from a deadlock-free operating guidelines of a maximal k-deadlock-free controller of service S by applying Definition 4.1 and Definition 4.3 respectively.

Definition 5.30 (Complete and compact canonical deadlock-free substitutes). Let $k \in \mathbb{N}$ be a message bound for each message channel. Let OG be an operating guideline that represents the set of all controllers of a maximal k-deadlock-free controller of service S, that is, $Comply_\beta(OG) = df_k Controllers(max_{df,k}(S))$.

Service $\mathcal{C}^2_{df,k}(S)$ as constructed from OG by applying Definition 4.1 is called a *complete canonical k-deadlock-free substitute* for service S.

Service $\widehat{\mathcal{C}}^2_{df,k}(S)$ as constructed from OG by applying Definition 4.3 is called a *compact canonical k-deadlock-free substitute* for service S.

The following lemma asserts that a complete canonical k-deadlock-free substitute $\mathcal{C}^2_{df,k}(S)$ for service S and a compact canonical k-deadlock-free substitute $\widehat{\mathcal{C}}^2_{df,k}(S)$ for service S are both solutions for a closure operator $\mathcal{B}max^2(S) = \mathcal{B}max(\mathcal{B}max(S))$ on S for deadlock freedom ($\mathcal{B} = df_k$).

Lemma 5.31 (Closure operation of k-deadlock-free substitutes). For each message bound $k \in \mathbb{N}$ and each k-deadlock-freely controllable service S:

$$S \;=_{df,k}\; \mathcal{C}^2_{df,k}(S) \;=_{df,k}\; \widehat{\mathcal{C}}^2_{df,k}(S).$$

Proof. Follows from Theorem 5.4 (for $\mathcal{B} = df_k$), Lemma 5.11, and Lemma 5.12. □

With the distinguished properties of a canonical deadlock-free controller, we can establish a coincidence between the set of all substitutes for a given service S under k-deadlock-free inclusion and the set of all services that refines a canonical k-deadlock-free substitute for S under stable failures. We illustrate this with Theorem 5.32.

Theorem 5.32 (Relationship with stable failures refinement and canonical deadlock-free substitute).
For each message bound $k \in \mathbb{N}$ and each k-deadlock-freely controllable service S :

$$df_k Inclusion(S) \;=\; f\text{-}refine(\mathcal{C}^2_{df,k}(S)) \;=\; f\text{-}refine(\widehat{\mathcal{C}}^2_{df,k}(S)).$$

Proof. Follows from Corollary 5.23, Theorem 4.38, and Lemma 4.31. □

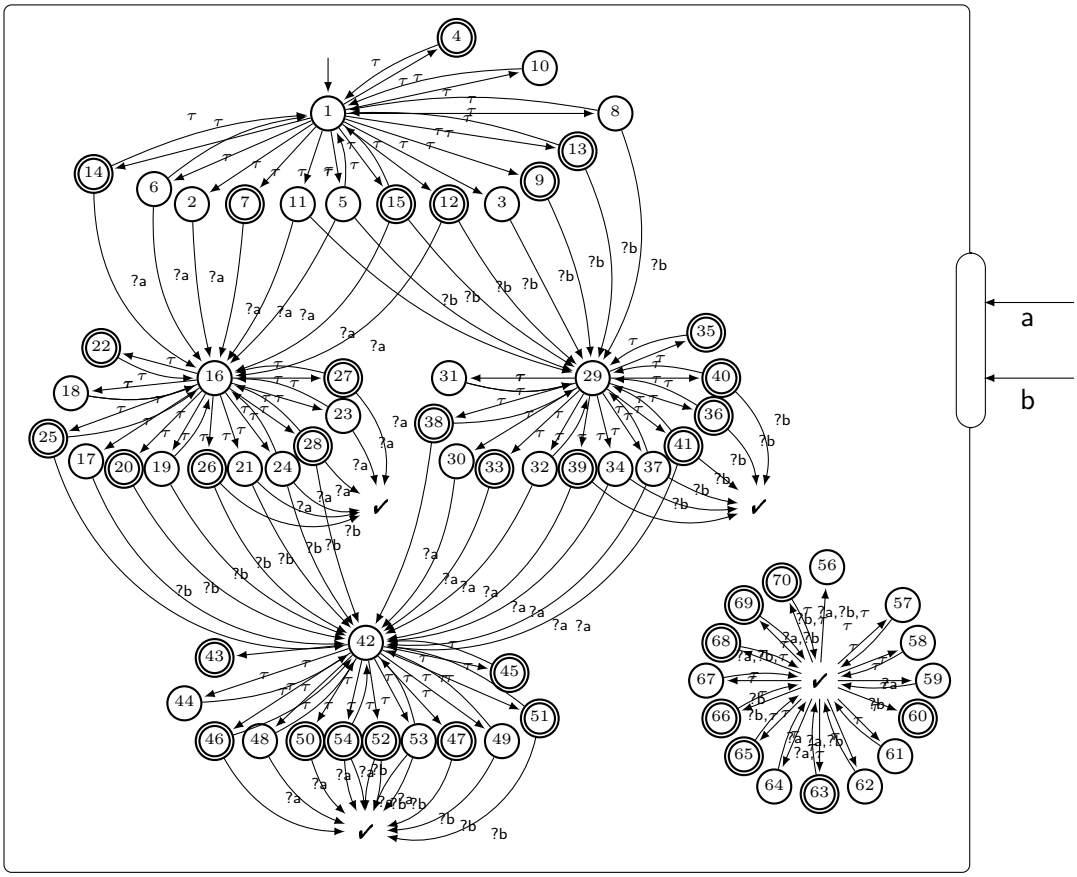


Figure 5.8.: A canonical k-deadlock-free substitute $\mathcal{C}^2_{df,k}(S_1)$ for service S_1 (from Figure 4.1) with $I = \{a, b\}$ and $O = \{\ \}$ for $k = 1$.

Example 5.33. Figure 5.8 illustrates a canonical k-deadlock-free substitute $\mathcal{C}^2_{df,k}(S_1)$ for service S_1 from Figure 4.1. Figure 5.9 illustrates a canonical k-deadlock-free substitute $\widehat{\mathcal{C}}^2_{df,k}(S_1)$ for service S_1. Each canonical service has an interface $I = \{a, b\}$ and $O = \{\ \}$, that is the same interface as service S_1, and is constructed from the operating guideline $mp_{df,k}(\mathcal{C}_{df,k}(S_1))^{\varphi}$ (from Figure 5.6) of a canonical k-deadlock-free controller $\mathcal{C}_{df,k}(S_1)$ (from Figure 4.2) of S_1. Both canonical services are equivalent under stable failures.

Consider services S_2, S_3, and S_4 from Figure 4.1. We see that each of them refines the service $\mathcal{C}^2_{df,k}(S_1)$ and the service $\widehat{\mathcal{C}}^2_{df,k}(S_1)$ under stable failures. With Theorem 5.32, we conclude that each of them is a substitute for S_1 under k-deadlock freedom inclusion. ◁

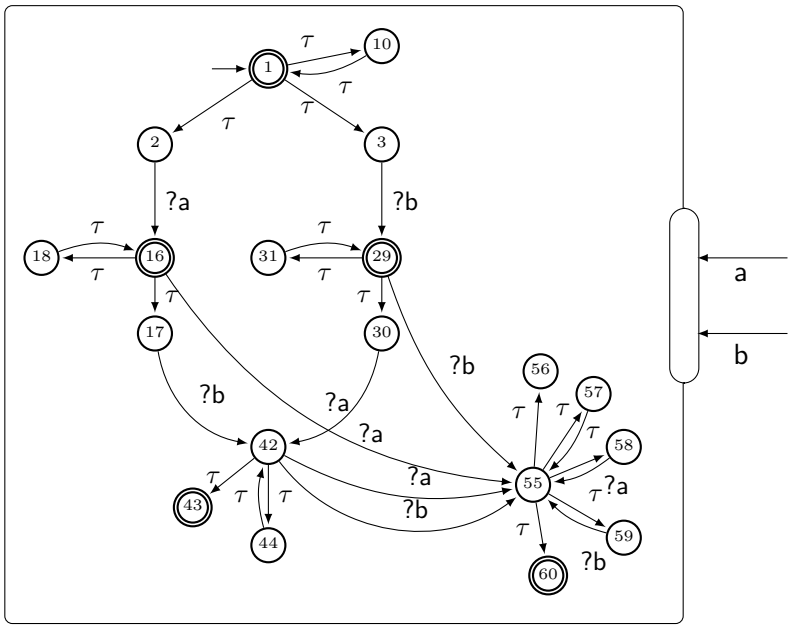


Figure 5.9.: A compact canonical k-deadlock-free substitute $\widehat{\mathcal{C}}^2_{df,k}(S_1)$ for service S_1 (from Figure 4.1) with $I = \{a, b\}$ and $O = \{\ \}$ for $k = 1$.

A canonical deadlock-free substitute for a given service equipped with the stable failures refinement relation is a natural candidate for representing all its substitutes under deadlock freedom. We discuss the respective applications of a canonical deadlock-free substitute as well as its optimized construction procedure in Chapter 7.

Canonical Responsive Substitute for a Service

In this section, we denote a *complete canonical k-responsive substitute* for service S by $\mathcal{C}^2_{rp,k}(S)$ and a *compact canonical k-responsive substitute* for service S by $\widehat{\mathcal{C}}^2_{rp,k}(S)$, each can be constructed from a responsive operating guideline of a maximal k-responsive controller of service S by applying Definition 4.6 and Definition 4.8 respectively.

Definition 5.34 (Complete and compact canonical responsive substitutes).
Let $k \in \mathbb{N}$ be a message bound for each message channel. Let OG be an operating guideline that represents the set of all controllers of a maximal k-responsive controller of service S, that is, $\mathit{Comply}_\beta(OG) = \mathit{df}_k\mathit{Controllers}(\mathit{max}_{rp,k}(S))$.

Service $\mathcal{C}^2_{rp,k}(S$ as constructed from OG by applying Definition 4.6 is called a *complete canonical k-responsive substitute* for service S.

Service $\widehat{\mathcal{C}}^2_{rp,k}(S)$ as constructed from OG by applying Definition 4.8 is called a *compact canonical k-responsive substitute* for service S.

The following lemma asserts that a complete canonical k-responsive substitute $\mathcal{C}^2_{rp,k}(S)$ for service S and a compact canonical k-responsive substitute $\widehat{\mathcal{C}}^2_{rp,k}(S)$ for service S are both solutions for a closure operator $\mathcal{B}max^2(S) = \mathcal{B}max(\mathcal{B}max(S))$ on S for responsiveness ($\mathcal{B} = rp_k$).

Lemma 5.35 (Closure operation of k-responsive substitutes).
For each message bound $k \in \mathbb{N}$ and each k-responsively controllable service S:

$$S \ =_{rp,k} \ \mathcal{C}^2_{rp,k}(S) \ =_{rp,k} \ \widehat{\mathcal{C}}^2_{rp,k}(S).$$

Proof. Follows from Theorem 5.4 (for $\mathcal{B} = rp_k$), Lemma 5.17, and Lemma 5.18. □

With distinguished properties of the canonical responsive controller, we can establish a coincidence between the set of all service that is a substitute for a given service S under k-responsive inclusion and the set of all services that refines a canonical k-responsive substitute for S under responsive failures. We illustrate this with Theorem 5.36.

Theorem 5.36 (Relationship with responsive failures refinement services and canonical responsive substitutes).
For each message bound $k \in \mathbb{N}$ and each k-responsively controllable service S :

$$rp_k Inclusion(S) \ = \ rf\text{-}refine(\mathcal{C}^2_{rp,k}(S)) \ = \ rf\text{-}refine(\widehat{\mathcal{C}}^2_{rp,k}(S)).$$

Proof. Follows from Theorem 5.7 (for $\mathcal{B} = rp_k$), Theorem 4.38, and Lemma 4.31. □

Example 5.37. Consider message bound $k = 1$. Figure 5.10 illustrates a complete canonical k-responsive substitute $\mathcal{C}^2_{rp,k}(S_1)$ for service S_1 from Figure 4.1. Figure 5.11 illustrates a compact canonical k-responsive substitute $\widehat{\mathcal{C}}^2_{rp,k}(S_1)$ for service S_1. Each canonical service has an interface, $I = \{a, b\}$ and $O = \{\ \}$, that is, same interface as service S_1, and is constructed from the operating guideline $mp_{rp,k}(\mathcal{C}_{rp,k}(S_1))^{\psi^*}$ (from Figure 5.7) of a canonical k-responsive controller $\mathcal{C}_{rp,k}(S_1)$ (from Figure 4.3) of service S_1. Both canonical services are equivalent under responsive failures.

Consider services S_2, S_3, and S_4 from Figure 4.1. We see that services S_2 and S_3 refine the service $\mathcal{C}^2_{rp,k}(S_1)$ and the service $\widehat{\mathcal{C}}^2_{rp,k}(S_1)$ under responsive failures. With Theorem 5.36, we conclude that each of them can is a substitute for service S_1 under k-responsiveness inclusion.

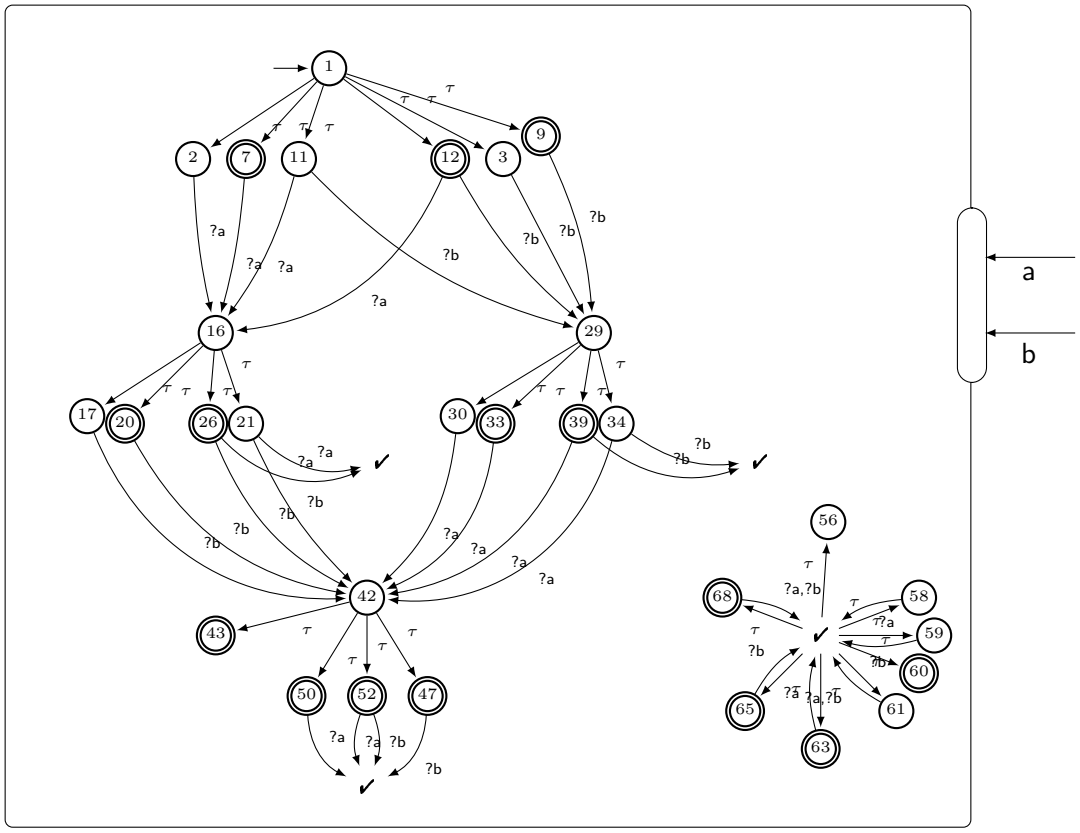


Figure 5.10.: A canonical k-responsive substitute $\mathcal{C}^2_{rp,k}(S_1)$ for service S_1 (from Figure 4.1) with $I = \{a, b\}$ and $O = \{\ \}$ for $k = 1$.

Nevertheless, service S_4 refines neither the service $\mathcal{C}^2_{rp,k}(S_1)$ nor the service $\widehat{\mathcal{C}}^2_{rp,k}(S_1)$ under responsive failures. This is due to state d_3 of S_4 that forms a stable τ-strongly connected component which refuses every event from $I \cup O \cup \{\mathit{final}\}$ and these events are not refused by a canonical substitute for S_1 after having executed an empty trace. Therefor, service S_4 is not a substitute for service S_1 under k-responsive inclusion. $\lhd$

Similarly to the canonical deadlock-free substitute, the canonical responsive substitute for a given service equipped with the responsive failures refinement relation is a natural candidate for representing all substitutes for the given service under responsiveness. We discuss the respective applications of a canonical responsive substitute in Chapter 7.

5.2.3. Experimental Results

In this section, we show the experimental results of constructing an operating guideline of a canonical responsive controller and constructing a canonical responsive substitute on the same set of services as presented in Section 4.1.3.

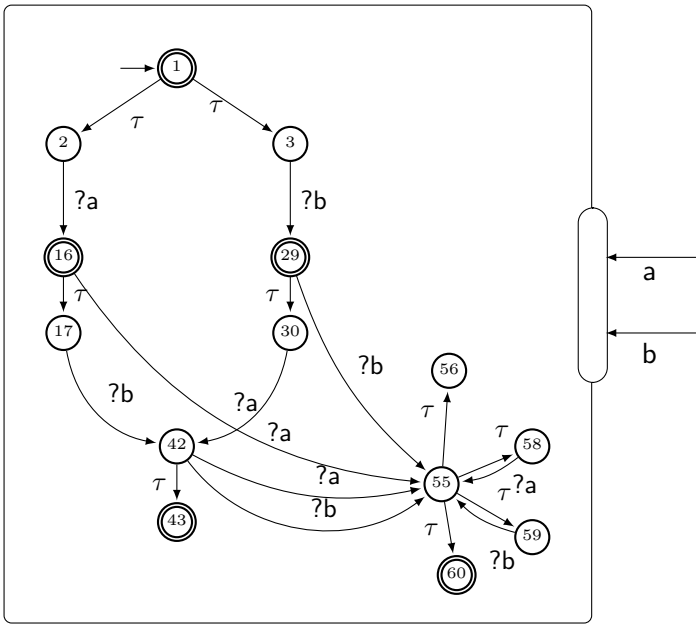


Figure 5.11.: A compact canonical k-responsive substitute $\widehat{\mathcal{C}}^2_{rp,k}(S_1)$ for service S_1 (from Figure 4.1) with $I = \{a, b\}$ and $O = \{\ \}$ for $k = 1$.

Illustrated in Table 5.1 and Table 5.2 are six service models. The three services A:*Identity Card issue*, B:*Car Analysis*, and C:*Product Delivery* are industrial service models provided by a company. Service D:*Purchase Order* is taken from the *WS-BPEL* specification [Alves et al., 2007]. Service F:*Product Reservation* is taken from our own library. These models were specified in the service description language *WS-BPEL*. Then we translated the models into specifications into *open nets* using the compiler *BPEL2oWFN* [Lohmann, 2007] (See also Figure 2.9). We also experimented with an open net model of the simple mail transfer protocol E:*SMTP* [Klensin, 2001].

With the open net models of these services, we employed the tool *Wendy* [Lohmann and Weinberg, 2010] to construct their operation guidelines. Then we employed our tool *Maxis* [Parnjai, 2011c] to construct a service automaton model of a compact canonical controller $\widehat{\mathcal{C}}_{rp,k}$ (illustrated in Table 4.2, Section 4.1.3) from an operating guideline. We translated a compact canonical controller $\widehat{\mathcal{C}}_{rp,k}$ into an open net model using the tool *PnAPI* [Mennicke et al., 2009]. Then we employed *Wendy* again to construct an operating guidelines from the open net model of compact canonical controller $\widehat{\mathcal{C}}_{rp,k}$.

Similar to Section 4.1.3, we illustrate all services in Table 5.1 (also in Table 5.2) as service automata in order to compare their sizes on the same scale. The size of a service is described in terms of the number of all states $|Q|$, the number of all final states $|\Omega|$, the number of all transitions $|\rightarrow|$, the number of τ transitions $|\stackrel{\tau}{\rightarrow}|$, the size of its input interface $|I|$, and the size of its output interface $|O|$. The size of the operating guideline OG is judged by the size of its underlying most permissive controller; the number of its states $|Q|$ and the number of its transitions $|\rightarrow|$.

Table 5.1.: Service S and an operating guideline of its compact canonical responsive controller $OG(\widehat{\mathcal{C}}_{rp,k}(S))$ for message bound $k = 1$

	Service S					$OG(\widehat{\mathcal{C}}_{rp,k}(S))$	
	$\lvert Q_S \rvert$	$\lvert \rightarrow_S \rvert$	$\lvert \xrightarrow{\tau}_S \rvert$	$\lvert I_S \rvert$	$\lvert O_S \rvert$	$\lvert Q \rvert$	$\lvert \rightarrow \rvert$
A : Identity Card Issue	14,569	71,332	66,500	2	9	1,537	9,739
B : Car Analysis	11,381	39,865	27,231	6	9	1,577	14,515
C : Product Delivery	4,148	13,832	9,288	5	9	1,377	11,870
D : Purchase Order	402	955	675	4	6	169	1,066
E : SMTP	60	92	0	9	5	791	8,279
F : Product Reservation	28	33	16	2	8	1,170	6,092

Table 5.2.: Comparison of services S and its canonical responsive substitute $\widehat{\mathcal{C}}^2_{rp,k}(S)$ for message bound $k = 1$

	Service S				$\widehat{\mathcal{C}}^2_{rp,k}(S)$			
	$\lvert Q_S \rvert$	$\lvert \Omega_S \rvert$	$\lvert \rightarrow_S \rvert$	$\lvert \xrightarrow{\tau}_S \rvert$	$\lvert Q \rvert$	$\lvert \Omega \rvert$	$\lvert \rightarrow \rvert$	$\lvert \xrightarrow{\tau} \rvert$
A	14,569	1	71,332	66,500	28,179	2	81,442	26,643
B	11,381	1	39,865	27,231	12,196	11	36,127	10,620
C	4,148	1	13,832	9,288	21,744	4	67,976	20,369
D	402	1	955	675	1,244	2	3,581	1,076
E	60	1	92	0	5,633	11	21,990	4,843
F	28	1	33	16	5,937	35	12,688	4,768

Illustrated in Table 5.1 is the comparison between services and the operating guidelines of their compact canonical responsive controllers. The results has shown that the size of the operating guideline is not necessarily in the smae order as the original service. Note, that we employed a compact canonical controller $\widehat{\mathcal{C}}_{rp,k}$ instead of a complete canonical controller $\mathcal{C}_{rp,k}$ to construct its operating guidelines due to its relatively smaller size (See Table 4.2). As the time and space complexity of operating guidelines construction is known to be exponential with respect to an increase of input size of a service [Massuthe, 2009, Stahl, 2009, Lohmann, 2010]; where the input size is described by the number of state, the size of input and output interfaces, and the message bound, the computation of each operating guideline on a complete canonical controller $\mathcal{C}_{rp,k}$ is undesirable due to its relatively very large size (See Table 4.2). Our experiment conducted on complete canonical controller $\mathcal{C}_{rp,k}$ are not shown in this thesis, because they were aborted while running either *Wendy* to compute operating guidelines or *PnAPI* to transform service automaton model into open net model, after hours of execution. Therefore, we chose a compact canonical responsive controller $\widehat{\mathcal{C}}_{rp,k}$ which is relatively compact in its size and is invariant under responsive substitution. In general, it is always practical to apply reduction techniques or reduction rules (for instance, transformation rules in Section 4.5) to reduce the input size of services before applying a tool whenever possible.

Table 5.2 shows the comparison between a service and its canonical responsive substitute

$\widehat{\mathcal{C}}^2_{rp,k}$. The results has shown that size of compact canonical responsive substitute $\widehat{\mathcal{C}}^2_{rp,k}$ does not change significantly in comparison to the size of the input service. As a service increases its size (from F to A), the size of its canonical substitute is not increasing in the same order. There are cases, for instance D:*Purchase Order*, E:*SMTP* and F:*Product Reservation*, where their canonical substitutes are relatively larger than the input service. This is because a canonical substitute of a given service structurally can represent the set of all substitutes of the input service. Nevertheless, the results has shown that, for the three industrial services; A:*Identity Card issue*, B:*Car Analysis*, and C:*Product Delivery*, the size of services and their canonical responsive substitutes are of the same order.

As the construction of an operating guideline OG of a canonical substitute $\mathcal{C}_{rp,k}$ (in Table 5.1) and a canonical substitute $\widehat{\mathcal{C}}^2_{rp,k}$ (in Table 5.2) from a given service requires a number several steps and therefore involves a pipeline of input and output to a number of tools, we employed our tool *Evans* [Parnjai, 2011b] to help performing all required steps in this section.

5.3. Minimal Controllers

In this section, we investigate a minimal $\mathcal{B}$-controller of a given service. A minimal $\mathcal{B}$-controller of service S is defined as a $\mathcal{B}$-controller of service S that is smaller than or equal to any other B-controller of S.

Definition 5.38 (Minimal $\mathcal{B}$-controller).
For each message bound $k \in \mathbb{N}$ and each compatibility property $\mathcal{B} = \{df_k, rp_k\}$, a minimal $\mathcal{B}$-controller of a $\mathcal{B}$-controllable service S is a service $\mathcal{B}min(S)$ that satisfies the following property:

$$(\forall P \in \mathcal{B}\text{-}Controllers(S) \ : \ \mathcal{B}min(S) \ \sqsubseteq_{\mathcal{B}} \ P).$$

Unlike a maximal $\mathcal{B}$-controller of a given service S, we cannot substitute $\mathcal{B}min(S)$ by any other $\mathcal{B}$-controller P of service S under inclusion. On the contrary, service $\mathcal{B}min(S)$ can substitute each $\mathcal{B}$-controller P of S under $\mathcal{B}$ inclusion. As the $\mathcal{B}$ inclusion relation $\sqsubseteq_{\mathcal{B}}$ is a preorder relation, $\mathcal{B}min(S)$ is in the $\sqsubseteq_{\mathcal{B}}$ preorder smaller than or equal to any other $\mathcal{B}$-controller P of service S.

In the following subsections, we first introduce a service with distinguished properties called *divergence service*, in Section 5.3.1. Then, we show in Section 5.3.2 that a divergence service is indeed a solution for a minimal k-deadlock-free controller of any given k-deadlock-freely controllable service. Finally in Section 5.3.3, we illustrate that, in general, there is no unique minimal k-responsive controller of a given k-responsively controllable service.

5.3.1. Divergence Service

In this section, we define a *divergence service* as a service that performs nothing but *diverges*, that is, it can only perform infinitely many unbroken steps of internal τ events.

We show that a divergence service is indeed a minimal k-deadlock-free controller of every k-deadlock-free controllable service.

Definition 5.39 (Divergence service).
A *divergence service* is a service automaton $[Q, q_0, I, O, \rightarrow, \Omega]$ in which every state $q \in Q$ is a divergent state, i. e., $enable(q) = \{\tau\}$.

A divergence service can perform an infinite unbroken sequence of internal τ events without sending message to or receiving message from any service that it interacts with, though its interface may not be empty. Note that a divergence service does not have a final state, as a divergence service also never offer a successfully terminating event *final* to its interacting service.

Example 5.40. Figure 5.12 shows four different divergent services, each service contains only divergent states. Each service has an arbitrary interface. ◁

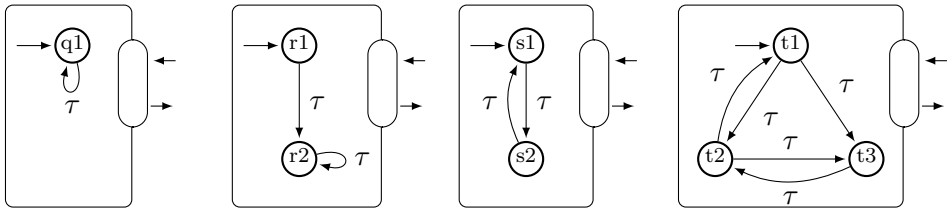


Figure 5.12.: Four divergent services.

Though never interacting with any service, a divergence service has a k-deadlock-free controller for each message bound $k \in \mathbb{N}$. We illustrate this with the following properties.

Property 5.41 For each message bound $k \in \mathbb{N}$ and each divergence service $S_\perp$ holds: $S_\perp$ is k-deadlock-freely controllable.

Proof. Suppose $S_\perp$ is not k-deadlock-freely controllable. For a k-deadlock-free controllable service P with an interface compatible to $S_\perp$, this implies, $\neg df_k(S_\perp \oplus P)$.

As every state q_S of $S_\perp$ is a divergent state, this means, for a state $[q_S, \mathcal{M}, q_P]$ in the composition $S_\perp \oplus P$ there is always an outgoing τ-transition. Let σ be a sequence of message events in P with $q_{0P} \stackrel{\sigma}{\Rightarrow} q_P$. As $\neg df_k(S_\perp \oplus P)$, this means that after P performing σ there must be a transition $q_P \stackrel{m}{\rightarrow} q'_P$ in P that violates the k-boundedness by sending message m that illegally updates $\mathcal{M}$ in $S_\perp \oplus P$. This implies that P can perform (possibly empty) σ before m without an interference from its partner. Therefore, it is not possible to deadlock-freely control P as P always violates the k-boundedness property. This contradicts to the assumption that P is k-deadlock-freely controllable.

Thus, there is a k-deadlock-free controllable P with interface compatible to $S_\perp$ such that $df_k(S_\perp \oplus P)$ holds. □

A divergence service is a service in which every state q is a divergent state. It is straightforward to see that a divergent state q can perform infinitely many steps of τ events but never performs other events. Intuitively, a divergence service never responds to its interacting service.

In the followings, we prove that a divergence service is not k-responsively controllable for any message bound $k \in \mathbb{N}$, as there is no other service with compatible interface that can interact responsively with a divergence service.

Property 5.42 For each message bound $k \in \mathbb{N}$ and each divergence service $S_\perp$ holds: $S_\perp$ is not k-responsively controllable.

Proof. Consider a k-responsively controllable service P with an interface compatible to $S_\perp$. Every state q_S of $S_\perp$ is a divergent state, i. e., $enable(q) = \{\tau\}$, and a divergent state is not a responsive state by definition. For a state $[q_S, \mathcal{M}, q_P]$ in the composition $S_\perp \oplus P$, service $S_\perp$ never updates the buffer $\mathcal{M}$ and does not offer any other event than an invisible τ event. This means that the behavior $Beh_{S_\perp \oplus P}(S_\perp)$ of $S_\perp$ with respect to the composition $S_\perp \oplus P$ is not responsive. Thus, it is not possible to responsively control divergent service $S_\perp$. □

5.3.2. Minimal Deadlock-free Controllers

In this section, we show that a divergence service defined in Section 5.3.1 is indeed a minimal k-deadlock-free controller of each k-deadlock-freely controllable service S for any message bound $k \in \mathbb{N}$.

Lemma 5.43 (Divergence service).
For each message bound $k \in \mathbb{N}$ and each service S and divergence service $P_\perp$ that is interface compatible to S holds:

$$(\forall P \in df_k Controllers(S) :: P_\perp \ \sqsubseteq_{df,k} P).$$

Proof. Consider a divergence service $P_\perp$. To prove that $P_\perp$ is smaller than or equal to every $P \in df_k Controllers(S)$, we show that (1) $P_\perp \in df_k Controllers(S)$ and (2) $df_k Controllers(P_\perp) \supseteq df_k Controllers(P)$.

1. Consider the composition $S \oplus_k P_\perp$. As every state q_P of P^* is a divergent state, this means, for a state $[q_S, \mathcal{M}, q_P]$ in $S \oplus_k P_\perp$ there is always an outgoing τ-transition. Therefore, $[q_S, \mathcal{M}, q_{P_\perp}]$ is not a deadlock state in $S \oplus_k P_\perp$. That is, $\neg df(S \oplus P^*)$ holds and $P_\perp \in df_k Controllers(S)$ follows.
2. Consider a service $P \in df_k Controllers(S)$.
   Suppose there is a service $T \in df_k Controllers(P)$ and $T \notin df_k Controllers(P_\perp)$. Hence, $df_k(T \oplus P)$ and $\neg df_k(T \oplus P_\perp)$ follows by definition. As $P_\perp$ is a divergent service, then $P_\perp$ is k-deadlock-free controllable according to Property 5.41.

As every state q_P of $P_\perp$ is a divergent state, this means that for a state $[q_T, \mathcal{M}, q_{P_\perp}]$ in $T \oplus P_\perp$, there is always an outgoing τ transition.
Let σ be a sequence of message events in T with $q_{0T} \stackrel{\sigma}{\Rightarrow} q_T$.
As $\neg df_k(T \oplus P_\perp)$ holds, this means that after performing σ there is a transition $q_T \stackrel{m}{\rightarrow} q'_T$ in T that violates the k-boundedness by sending message m that illegally updates $\mathcal{M}$ in $T \oplus P_\perp$. This implies that T can perform (possibly empty) σ before m without an interference from its partner.
Similar for the composition $T \oplus P$, in which T can perform (possibly empty) σ before m without an interference from its partner. This means, the composition $T \oplus P$ also violates the k boundedness and this contradicts to $df_k(T \oplus P)$.
Therefore, $df_k Controllers(P_\perp) \supseteq df_k Controllers(P)$ holds.

Thus, the lemma holds. □

It is possible to derive a minimal deadlock-free controller of S from a canonical deadlock-free controller of S by applying transformation rules such as $df(1)$ (cf. Definition 4.64) and $F(1)$ (cf. Definition 4.68). Though the two rules guarantee to preserve stable failures refinement under transformation, but our result from Lemma 4.34 has indicated a coincidence in one direction. As every node of a most permissive k-deadlock-free controller of S contains a τ-loop and $\{\tau\}$ is always one of the valid canonical choices of the most-permissive deadlock-free controller, hence the canonical deadlock-free controller always has a choice to perform divergent behavior by construction. Illustrated in Figure 4.9 is an application of rule $F(1)$ on M, assuming M is a canonical k-deadlock free controller of a given service. The application transforms service M into a divergent service O_1. A divergent service O_1 can be transformed into divergent service O_2 by applying rule $F(1)$.

5.3.3. Minimal Responsive Controllers

In this section, we show that there is, in general, no unique minimal element in the responsive preordered set of services.

Lemma 5.44 (k-responsive controller is not unique).
In general, there is no unique minimal k-responsive controller of a k-responsively controllable service S for each mesage bound $k \in \mathbb{N}$.

Proof. Let $rp_k Controllers(S) = Comply_\gamma(rp\text{-}mp(S)^{\psi^*})$ and $\mathcal{C}_{rp,k}(S)$ be a canonical k-responsive controller of service S constructed from $rp\text{-}mp(S)^{\psi^*}$.

Let $ch_1, ch_2 \in \psi^*(q)$ be two choices at state q of $\mathcal{C}_{rp,k}(S)$. Clearly, $ch_1 \neq \{\tau\}$ and $ch_2 \neq \{\tau\}$ as $\{\tau\}$ is not a valid choice for any k-responsive controller. Suppose $ch_1 \cap ch_2 = \emptyset$.

Let P_1 be a service that is transformed from $\mathcal{C}_{rp,k}(S)$ be applying rules $rp(1)$ followed by $F(1)$ multiple times to remove τ until it is not possible to remove τ from P_1 by applying rule $F(1)$. Assume P_1 preserve choice ch_1 under transformation. Let P_2 be a service that is transformed from $\mathcal{C}_{rp,k}(S)$ be applying rules $RF(1)$ followed by $F(1)$

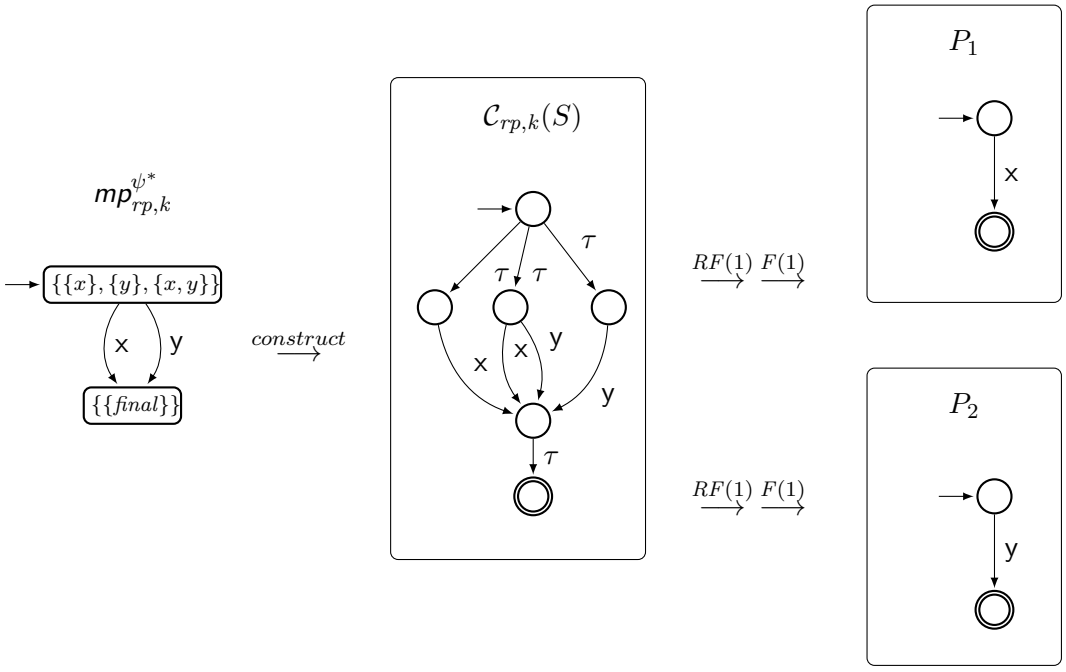


Figure 5.13.: Illustration for Lemma 5.44.

multiple times to remove τ until it is not possible to remove τ from P_2 by applying rule $F(1)$. Assume P_2 preserve choice ch_2 under transformation (for instance, see Figure 5.13 where $ch_1 = \{x\}$ and $ch_2 = \{y\}$).

Application of rules $RF(1)$ and $F(1)$ guarantees that each of P_1 and P_2 is indeed smaller than $\mathcal{C}_{rp,k}(S)$. From Figure 5.13, it is straightforward to see that there exists a k-responsive controller of P_1 that is not a k-responsive controller of P_2 and vice versa. This implies that it is not necessarily the case that P_1 and P_2 are k-responsively equivalent.

Thus, we conclude that in general there is no common minimal k-responsive controller for a k-responsively controllable service S. □

5.4. Concluding Remarks

In this chapter, we studied a relationship between a service and the set of all its $\mathcal{B}$-controllers under two behavioral compatibility criteria $\mathcal{B} \in \{df_k, rp_k\}$. We defined a *maximal $\mathcal{B}$-controller* of a given service as its $\mathcal{B}$-controller that is in the $\mathcal{B}$-preorder larger than or equal to any other $\mathcal{B}$-controller of the given service. As a maximal element of the $\mathcal{B}$-preordered set of controllers, a maximal $\mathcal{B}$-controller exhibits several distinguished properties. Two of these properties are the characterization of all substitutes for a given service under $\mathcal{B}$-inclusion and a closure operation on the set of services.

In practice, a solution for a maximal $\mathcal{B}$-controller from a given service is commendatory. For each behavioral compatibility criterion $\mathcal{B}$, we proposed two canonical $\mathcal{B}$-controllers

of the given service as previously introduced in Chapter 4, and illustrated that each controller is indeed a solution for a maximal $\mathcal{B}$-controller of the given service. Though dissimilar in size, each controller can be constructed from a given service by employing operating guidelines technique that represents the set of all $\mathcal{B}$-controllers of a given service. With the constructed canonical $\mathcal{B}$-controller, we can construct a finite representation of the set all substitution services under $\mathcal{B}$-inclusion as well as synthesize a canonical representative of the set. Naturally, we can also construct a more compact canonical representative of the given service by employing similar procedure as for constructing its canonical $\mathcal{B}$-controller. These results enable several interesting analysis and synthesis tasks on service substitution. We present experimental results and discuss further the related applications in Chapter 7.

For the sake of its own interest, we also studied a *minimal $\mathcal{B}$-controller* of a given service for each behavioral compatibility criterion $\mathcal{B}$. In contrast to a maximal $\mathcal{B}$-controller, a minimal $\mathcal{B}$-controller of a given service is a $\mathcal{B}$-controller that is in the $\mathcal{B}$-preorder smaller than or equal to any other $\mathcal{B}$-controller of the given service. This way, we can substitute any $\mathcal{B}$-controller of a given service by its minimal $\mathcal{B}$-controller, but not the other way round. In case of deadlock freedom $\mathcal{B} = df_k$, there exists a unique class of services, called *divergence service*, that is a common minimal k-deadlock-freedom controller of every k-deadlock-freely controllable service. In case of responsiveness $\mathcal{B} = df_k$, there exists no unique class of a minimal k-responsive controller that is common among all k-responsively controllable services, however.

In the next chapter, we will investigate an equivalence relation of services and characterize the set of all services that are equivalent to a given service under a given behavioral compatibility criterion $\mathcal{B}$. One major goal is to employ such a characterization to synthesize from a given service its equivalent service, which preserves exactly the same set of $\mathcal{B}$-controllers, yet relatively smaller in size than its canonical representatives and (possibly) than the given service itself.

6. Characterization of all Equivalent Substitutes

In this chapter, we study an equivalence relation on situations of services and investigate how a complete canonical substitution service as introduced in the previous chapter behaves with respect to the set of its controllers. We employ the equivalence relation on situations of the complete canonical substitute for a given service to characterize the set of all services that are equivalent to a given service under a given behavioral compatibility criterion.

The chapter is organized as follows. In Section 6.1, we define an equivalence relation on situations of services. Based on the equivalence relation, we characterize in Section 6.2 the set of all equivalent substitutes for a given service under deadlock freedom. In Section 6.3, we characterize the set of all equivalent substitutes for a given service under responsiveness. Finally, Section 6.4 concludes the chapter.

6.1. Equivalence of Situations

In this section, we study an equivalence relation on situations of services. As introduced in Section 3.3.1, the concept of *situations* is crucial for studying the interacting behavior between a service and its controller. Informally, a situation is a snapshot of the service composition from the viewpoint of one service, consisting of its own state and the state of the message buffer.

Consider the composition of two interface compatible service automata S and P. Recall from Section 2.2, we model each state q of the composition $S \oplus P$ by a tuple $q = [q_S, \mathcal{M}, q_P]$ consisting of a state q_S of S, a multiset $\mathcal{M}$ of all messages currently pending in the buffer between S and P, and a state q_P of P. Recall from Section 3.3.1, a situation of service S consists of its own state q_S and a multiset $\mathcal{M}$ of all messages currently pending in the buffer. A knowledge of P at state q_P about S is the mapping $\mathcal{K}(q_P)$ from state q_P to all situations of S that involve state q_P.

Two situations of service S that involve the same state q_P may or may not influence the choices at state q_P in a similar manner. Intuitively, two situations of a given service are equivalent whenever they similarly influence behavior of its interacting service. We define an equivalence relation of two situations with respect to a set E of events of service P as follows.

Definition 6.1 (Equivalence of situations).
Let S and P be two interface compatible service automata. Let q_P be a state of P and $[q, \mathcal{M}]$ and $[q', \mathcal{M}']$ be two situations of S at state q_P and $E \subseteq ?\Sigma_P \cup !\Sigma_P \cup \{final, \tau\}$ be a set of events of P. Then, the equivalence relation between two situations $[q, \mathcal{M}]$ and $[q', \mathcal{M}']$ with respect to the set E, is defined as

$$[q, \mathcal{M}] \sim_E [q', \mathcal{M}'] \;\Leftrightarrow\; (\forall e \in E :: activ_k([q, \mathcal{M}], e) \Leftrightarrow activ_k([q', \mathcal{M}'], e)).$$

for a message bound $k \in \mathbb{N}$ on each message channel.

Recall Definition 3.39, the predicate $activ_k$ indicates whether an event e can be performed on the given situation $[q, \mathcal{M}]$ without violating the message bound k on message channels. Two equivalent situations $[q, \mathcal{M}]$ and $[q', \mathcal{M}']$ with respect to the set E of events have the property that they are activated similarly by each event e in E. Intuitively, this means that one equivalent situation can act on behalf of another without affecting a combination of events in the set E.

Based on the equivalence relation of two situations, we define an equivalence class and partition of situations.

Definition 6.2 (Equivalence class, partition of equivalent situations).
Let S and P be two interface compatible service automata. Let $\mathcal{K}$ be a set of situations of S that contains a situation $[q, \mathcal{M}]$ of S. Let E be a set of events of P. Then, the

equivalence class of a situation $[q, \mathcal{M}]$ in $\mathcal{K}$ with respect to E is the set of all situations in $\mathcal{K}$ which are equivalent to $[q, \mathcal{M}]$ with respect to E, i. e.,

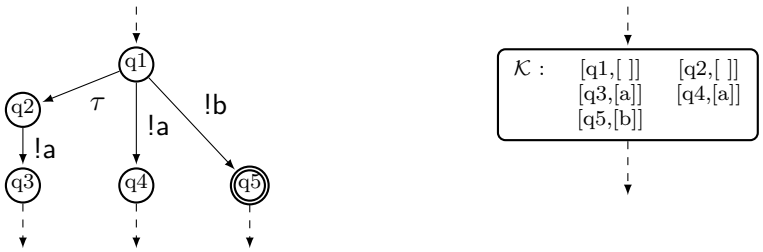
$$\{[q, \mathcal{M}]\}_{\mathcal{K},E} = \{[q', \mathcal{M}'] \in \mathcal{K} \mid [q, \mathcal{M}] \sim_E [q', \mathcal{M}']\}.$$

The set of all equivalence classes in $\mathcal{K}$ forms an *equivalence partition* of $\mathcal{K}$ with respect to E, denoted by $\xi_E(\mathcal{K})$.

As the set of situations of a given service is finite (due to the assumption of a message bound k on each message channel), there is a finite number of equivalence classes of situations. Note, that an empty set $\mathcal{K}$ of situations yields an empty partition $\xi_E(\mathcal{K})$, that is, $\mathcal{K} = \emptyset \Leftrightarrow \xi_E(\mathcal{K}) = \emptyset$, whereas an empty set E of events yields an equivalent partition of $\mathcal{K}$, that is, $\xi_E(\mathcal{K}) = \{\mathcal{K}\}$.

The following example shows the calculation of an equivalence partition of the set of situations with respect to a given set of events.

Example 6.3. The following figure shows a fragment of a service automaton S on the left hand side and its related situations in $\mathcal{K}$ on the right hand side.



The knowledge $\mathcal{K}$ contains five situations of service S. Consider message bound $k = 1$ on each message channel and a given set E of events with $E = \{?a, ?b, \tau, \mathit{final}\}$. The equivalence of situations in $\mathcal{K}$ with respect to E is illustrated in the following table.

$[q, \mathcal{M}] \in \mathcal{K}$	$[q1, [\,]]$	$[q2, [\,]]$	$[q3, [a]]$	$[q4, [a]]$	$[q5, [b]]$
$activ_k([q, \mathcal{M}], ?a)$	*false*	*false*	*true*	*true*	*false*
$activ_k([q, \mathcal{M}], ?b)$	*false*	*false*	*false*	*false*	*true*
$activ_k([q, \mathcal{M}], \tau)$	*true*	*true*	*true*	*true*	*true*
$activ_k([q, \mathcal{M}], \mathit{final})$	*false*	*false*	*false*	*false*	*true*

The two situations $[q1, [\,]]$ and $[q2, [\,]]$ are equivalent with respect to E, because each $e \in E$ activates each situation in a similar way as the other situation. The two situations $[q3, [a]]$ and $[q4, [a]]$ are also equivalent with respect to E whereas none of the other situations in $\mathcal{K}$ is equivalent to $[q5, [b]]$ with respect to E. Therefore, the equivalence partition of $\mathcal{K}$ with respect to E is $\xi_E(\mathcal{K}) = \{\ \{[q1, [\,]], [q2, [\,]]\},\ \{[q3, [a]], [q4, [a]]\},\ \{[q5, [b]]\}\ \}$. ◁

In the two subsequent sections, we will employ the equivalence of situations to investigate the circumstances in which two services behaves similarly with respect to their controllers. Note, that the two sections share common techniques for characterizing all equivalent substitutes for a given service, though the underlying artifacts are significantly different.

6.2. Characterization of Deadlock-freely Equivalent Substitutes

Two services are equivalent under deadlock freedom whenever they have the same set of deadlock-free controllers. Nevertheless, there is a case where one service can have a order of events that is completely different from the one of another service in the same deadlock-free equivalence class. In Section 5.2.2, we introduced a complete canonical deadlock-free service of a given service as a service that is equivalent to the given service under deadlock freedom. We have shown that a complete canonical deadlock-free service is structurally more liberal than any other service in the same deadlock-free equivalence class in the sense that it encodes all possible sequences and choices of events within its own structure.

In this section, we study a complete canonical deadlock-free service of a given service and investigate how it influences all its deadlock-free controllers. For this purpose, we first define in Section 6.2.1 canonical stable situations as representative of situations that can influence its controllers. In Section 6.2.2, we employ the equivalence relation on situations to partition the relevant canonical stable situations and calculate the sets of *covering situations* of the complete canonical deadlock-free service that is associated with the calculated partition. In Section 6.2.3, we define a matching relation of covering situations and propose one necessary condition for deciding deadlock freedom based on the matching relation. Finally, we present in Section 6.2.4 a deadlock-free equivalence guideline for characterizing the set of all services that are equivalent to a given service under deadlock freedom.

6.2.1. Canonical Stable Situations

In this section, we study a situation of a given service that influences how its deadlock-free controllers behave. We first recall from Section 3.3.2 a *stable situation* as a situation in which a service *cannot perform an event* without any help, such as input message, from its interacting partner. To interact with the service in a deadlock-free manner, a controller must offer a choice that can resolve the stable situation. Different controller may offer different choices to resolve the same situation. Nevertheless, there are some stable situations which are more influential to its controllers than other stable situations, as every controller *must* provide the same choice that can resolve such a situation in a deadlock-free manner.

For this purpose, we define a *canonical stable situation* as a stable situation of a given service that can influence exactly the choices that every controller of the service must provide.

Definition 6.4 (Canonical stable situations).
Let $\mathcal{K}(q)$ be the knowledge at state q of a service P about its interface compatible service S. Let $\widehat{\varphi}(q)$ be the set of all canonical deadlock-free choices at state q. Let $\widehat{\Sigma}_{\varphi}(q)$ be the set of all events that are described by $\widehat{\varphi}(q)$ and $\mathcal{L}_k(\mathbb{E}_f, \mathcal{K}(q))$ be the set of all *legally updated events* of $\mathcal{K}(q)$.

Then, the stable situation $[q, \mathcal{M}]$ is a *canonical stable situation* at state q if for each event $e \in \mathcal{L}_k(\mathbb{E}_f, \mathcal{K}(q))$ and $e \notin \widehat{\Sigma}_{\varphi}(q)$ holds: $\mathit{activ}_k([q, \mathcal{M}], e) = \mathit{false}$.

The set of all canonical stable situations at state q of P is denoted by $\widehat{\mathcal{K}}_{\varphi}(q)$.

Recall from Chapter 3, that a canonical deadlock-free choice $\widehat{\varphi}(q)$ at state q of a given service P is a choice that the service must provide in order to resolve the situations of its interacting partner (e. g., service S) in a deadlock-free manner (cf. Definition 3.27). A canonical deadlock-free event $\widehat{\Sigma}_{\varphi}(q)$ at state q is an event that is described by a canonical deadlock-free choice at state q (cf. Definition 3.28).

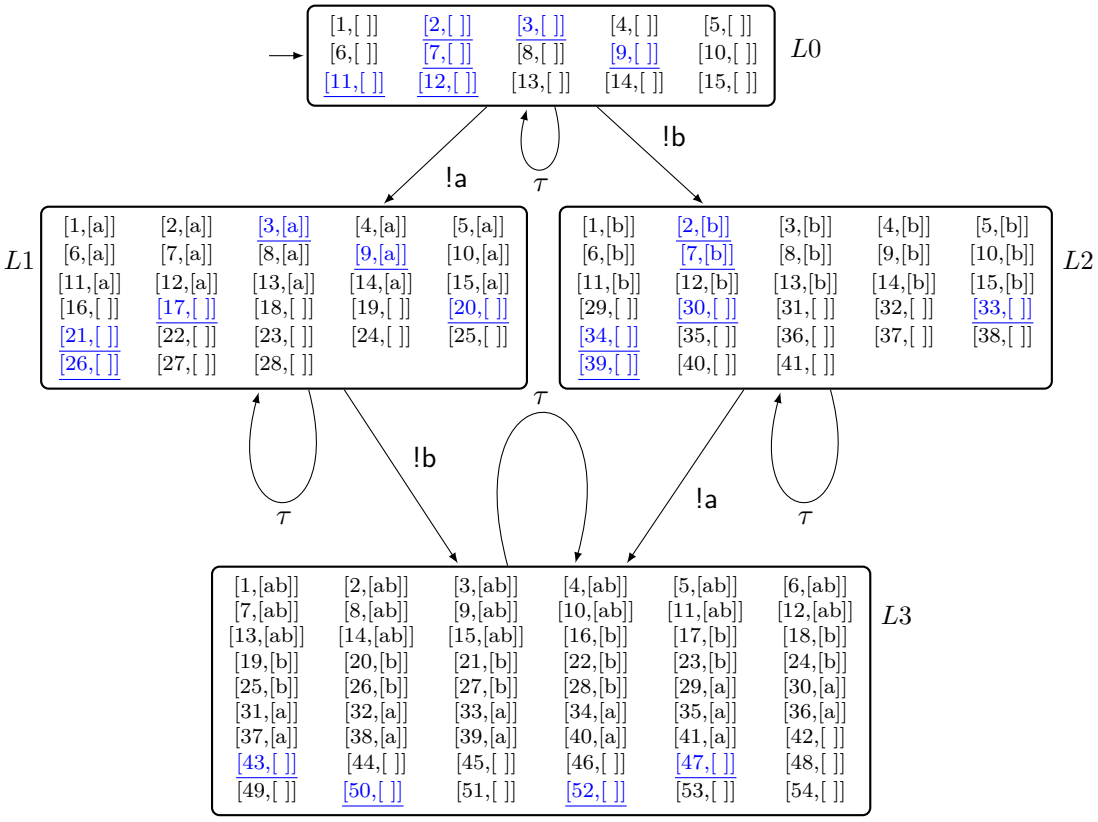
Given a set $\mathcal{K}^* = \mathit{stable}(\mathcal{K}(q))$ of stable situations of S at state q of P, a *canonical stable situation* of $\mathcal{K}^*$ is a situation of S that can only be updated legally by a canonical deadlock-free event of P, not by any other event of P. A set of all *legally updated events* $\mathcal{L}_k(\mathbb{E}_f, \mathcal{K}(q))$ of $\mathcal{K}(q)$ is the set of all events which never yield a situation that violates the message bound k for some message channel from the given set $\mathcal{K}(q)$ of situations (cf. Definition 3.47). To this respect, the calculation of canonical stable situations does not take into account an event e that illegally updates situations $\mathcal{K}(q)$ at state q. Such an event definitely violates the k-boundedness property of the service composition.

The following example shows the canonical stable situations of a complete canonical k-deadlock-free service of a given service when interacting with its most permissive k-deadlock-free controller.

Example 6.5. Consider service S_1 from Figure 4.1 and its complete canonical k-deadlock-free service $S_1^* = \mathcal{C}^2_{df,k}(S_1)$ from Figure 5.8 for message bound $k = 1$. Illustrated in Figure 6.1 is a most permissive k-deadlock-free controller $mp_{df,k}(S_1^*)$ of S_1^* for $k = 1$.

The controller $mp_{df,k}(S_1^*)$ consists of four states; $L0$, $L1$, $L2$, and $L3$. Each state q of the controller $mp_{df,k}(S_1^*)$ represents its knowledge of all relevant situations of S_1^* that are associated with state q in the composition $S_1^* \oplus mp_{df,k}(S_1^*)$ of S_1^* and $mp_{df,k}(S_1^*)$. The underlined situations denote the stable situations of S_1^*.

The most permissive k-deadlock-free controller $mp_{df,k}(S_1^*)$ is bisimilar to a most permissive k-deadlock-free controller $mp_{df,k}(S_1)$ of S_1 from Figure 4.2. This is because S_1^* is equivalent to its original service S_1 from Figure 4.1 under deadlock freedom (asserted by Lemma 5.31). The set of canonical deadlock-free choices of the two controllers $mp_{df,k}(S_1^*)$ and $mp_{df,k}(S_1)$ at each pair of bisimilar states is also the same set (asserted by Corollary 3.56). Though the situations of S_1^* associated with the controller $mp_{df,k}(S_1^*)$ are different than situations of S_1 associated with the controller $mp_{df,k}(S_1)$ (not shown here). By construction, the structure of S_1^* encodes all possible sequences and choices of events within its own structure.



$$
\begin{aligned}
\widehat{\varphi}(L0) &= \{\ \{!a\},\{!b\},\{\tau\}\ \}\\
\widehat{\varphi}(L1) &= \{\ \{!a\},\{\tau\}\ \}\\
\widehat{\varphi}(L2) &= \{\ \{!b\},\{\tau\}\ \}\\
\widehat{\varphi}(L3) &= \{\ \{\mathit{final}\},\{\tau\}\ \}
\end{aligned}
$$

Figure 6.1.: A most permissive k-deadlock-free controller $mp_{df,k}(S_1^*)$ of a complete canonical k-deadlock-free service $S_1^* = \mathcal{C}^2_{df,k}(S_1))$ from Figure 5.8 for message bound $k = 1$. Each state of the controller contains knowledge of $S_1^* = \mathcal{C}^2_{df,k}(S_1))$ and is equipped with a set of canonical deadlock free choices.

$[q,\mathcal{M}] \in \mathit{stable}(\mathcal{K}(L0))$	[2, [ ]]	[3, [ ]]	[7, [ ]]	[9, [ ]]	[11, [ ]]	[12, [ ]]
$\mathit{activ}_k([q,\mathcal{M}],!a)$	*true*	*true*	*true*	*true*	*true*	*true*
$\mathit{activ}_k([q,\mathcal{M}],!b)$	*true*	*true*	*true*	*true*	*true*	*true*
$\mathit{activ}_k([q,\mathcal{M}],\tau)$	*true*	*true*	*true*	*true*	*true*	*true*
$\mathit{activ}_k([q,\mathcal{M}],\mathit{final})$	*false*	*false*	*true*	*true*	*false*	*true*
$\mathit{enable}(q)$	$\{?a\}$	$\{?b\}$	$\{?a,\mathit{final}\}$	$\{?b,\mathit{final}\}$	$\{?a,?b\}$	$\{\ ?a,?b,\mathit{final}\}$

At state $L0$ of $mp_{df,k}(S_1^*)$, the set of all canonical choices is $\widehat{\varphi}(L0) = \{\{!a\},\{!b\},\{\tau\}\}$ and the set of all canonical events is $\widehat{\Sigma}_\varphi(L0) = \{!a,!b,\tau\}$. To calculate $\widehat{\mathcal{K}}(L0)$, we consider the activation of events for all stable situations at $L0$ in the following table.

The three stable situations $[7, [\ ]]$, $[9, [\ ]]$, and $[12, [\ ]]$ are not canonical situations of S_1^*. This is because there exists event $\mathit{final} \notin \widehat{\Sigma}_\varphi(L0)$ and $\mathit{activ}_k([7, [\ ]], \mathit{final}) = \mathit{activ}_k([9, [\ ]], \mathit{final}) = \mathit{activ}_k([12, [\ ]], \mathit{final}) = \mathit{true}$. Therefore, $\widehat{\mathcal{K}}(L0) = \{[2, [\ ]], [3, [\ ]], [11, [\ ]]\}$.

$[q, \mathcal{M}] \in \mathit{stable}(\mathcal{K}(L1))$	[3, [a]]	[9, [a]]	[17, [ ]]	[20, [ ]]	[21, [ ]]	[26, [ ]]
$\mathit{activ}_k([q, \mathcal{M}], !a)$	*false*	*false*	*true*	*true*	*true*	*true*
$\mathit{activ}_k([q, \mathcal{M}], !b)$	*true*	*true*	*true*	*true*	*true*	*true*
$\mathit{activ}_k([q, \mathcal{M}], \tau)$	*true*	*true*	*true*	*true*	*true*	*true*
$\mathit{activ}_k([q, \mathcal{M}], \mathit{final})$	*false*	*true*	*false*	*true*	*false*	*true*
$\mathit{enable}(q)$	$\{?b\}$	$\{?b, \mathit{final}\}$	$\{?b\}$	$\{?b, \mathit{final}\}$	$\{?a, ?b\}$	$\{?a, ?b, \mathit{final}\}$

At state $L1$ of $mp_{df,k}(S_1^*)$, the set of all canonical choices is $\widehat{\varphi}(L1) = \{\{!b\}, \{\tau\}\}$ and the set of all canonical events is $\widehat{\Sigma}_\varphi(L1) = \{!b, \tau\}$. To calculate $\widehat{\mathcal{K}}(L1)$, we consider the activation of each event for stable situations at $L1$ in the following table.

The three stable situations $[9, [a]]$, $[20, [\ ]]$, and $[26, [\ ]]$ are not canonical situations of S_1^*. This is because there exists event $\mathit{final} \notin \widehat{\Sigma}_\varphi(L1)$ and $\mathit{activ}_k([9, [a]], \mathit{final}) = \mathit{activ}_k([20, [\ ]], \mathit{final}) = \mathit{activ}_k([26, [\ ]], \mathit{final}) = \mathit{true}$. The stable situation $[21, [\ ]]$ is a canonical situation, though $\mathit{activ}_k([21, [\ ]], !a) = \mathit{true}$, but $!a$ is not a canonical event in $\widehat{\Sigma}_\varphi(L1)$. Event $!a$ cannot legally update $L1$ for $k = 1$ due to a situation such as $[3, [a]]$ in $L1$. Therefore, $\widehat{\mathcal{K}}(L1) = \{[3, [a]], [17, [\ ]]\}$.

$[q, \mathcal{M}] \in \mathit{stable}(\mathcal{K}(L3))$	[43, [ ]]	[47, [ ]]	[50, [ ]]	[52, [ ]]
$\mathit{activ}_k([q, \mathcal{M}], !a)$	*true*	*true*	*true*	*true*
$\mathit{activ}_k([q, \mathcal{M}], !b)$	*true*	*true*	*true*	*true*
$\mathit{activ}_k([q, \mathcal{M}], \tau)$	*true*	*true*	*true*	*true*
$\mathit{activ}_k([q, \mathcal{M}], \mathit{final})$	*true*	*true*	*true*	*true*
$\mathit{enable}(q)$	$\{\mathit{final}\}$	$\{?b, \mathit{final}\}$	$\{?a, \mathit{final}\}$	$\{?a, ?b, \mathit{final}\}$

At state $L3$ of $mp_{df,k}(S_1^*)$, the set of all canonical choices is $\widehat{\varphi}(L3) = \{\{\mathit{final}\}, \{\tau\}\}$ and the set of all canonical events is $\widehat{\Sigma}_\varphi(L3) = \{\mathit{final}, \tau\}$. To calculate $\widehat{\mathcal{K}}(L3)$, we consider the activation of each event for stable situations at $L3$ in the following table.

At state $L3$, every stable situation is a canonical situation. This is because there exists event $\mathit{final} \in \widehat{\Sigma}_\varphi(L3)$ with $\mathit{activ}_k([20, [\ ]], \mathit{final}) = \mathit{activ}_k([26, [\ ]], \mathit{final}) = \mathit{true}$. Though $!a$ and $!b$ are not a canonical event in $\widehat{\Sigma}_\varphi(L3)$ and each event can activate every stable situation of $L3$. However, both $!a$ and $!b$ are not legally updated events of $L3$ due to other situations in $L3$ such as $[1, [ab]]$ where either $!a$ or $!b$ will violate message bound $k = 1$ on message channel a and b respectively. Therefore, $\widehat{\mathcal{K}}(L3) = \{[43, [\ ]], [47, [\ ]], [50, [\ ]], [52, [\ ]]\}$.

The canonical stable situations of S_1^* from Figure 5.8 are illustrated as follows:

$L0:\ \widehat{\Sigma}_\varphi(L0) = \{!a, !b, \tau\}$ and $\widehat{\mathcal{K}}(L0) = \{[2, [\ ]], [3, [\ ]], [11, [\ ]]\}$,

$L1:\ \widehat{\Sigma}_\varphi(L1) = \{!b, \tau\}$ and $\widehat{\mathcal{K}}(L1) = \{[3, [a]], [17, [\ ]]\}$,

$L2:\ \widehat{\Sigma}_{\varphi}(L2) = \{!a, \tau\}$ and $\widehat{\mathcal{K}}(L2) = \{[2, [b]],\ [30, [\ ]]\}$, and

$L3:\ \widehat{\Sigma}_{\varphi}(L3) = \{\mathit{final}, \tau\}$ and $\widehat{\mathcal{K}}(L3) = \{[43, [\ ]],\ [47, [\ ]],\ [50, [\ ]],\ [52, [\ ]]\}$.

In this example, the calculation of $\widehat{\mathcal{K}}(L2)$ is omitted as it is almost similar to the calculation of $\widehat{\mathcal{K}}(L1)$. ◁

6.2.2. Deadlock-free Covering Situations

In the previous section, we introduced the canonical stable situations as situations which can influence exactly the canonical deadlock-free choices; these are the choices which every deadlock-free controller the given service must provide. Inherently not every canonical stable situation has an influence on the canonical choices in a similar way. With respect to a set of events described by the canonical stable choices, two equivalent canonical situations intuitively mean that each situation can act on behalf of the other without affecting a canonical choice of a controller.

In this section, we employ the equivalence relation on situations as defined in Section 6.1 to calculate the set of *deadlock-free covering situations* of a given service S^*. For each state q of a most permissive deadlock-free controller of S^*, we employ the equivalence relation to partition the relevant canonical stable situations of S^* with respect to the set of events described by canonical choices at q. Then we calculate the set of *deadlock-free covering situations* of S^* by performing a union operation on all subsets of all partitions at all states of a most permissive deadlock-free controller of S^*.

We define the set $Cov_{\varphi}(S^*)$ of *deadlock-free covering situations* of service S^* as follows.

Definition 6.6 (Deadlock-free covering situations).
Let $mp_{df,k}(S^*)$ be a most permissive k-deadlock-free controller of a k-deadlock-freely controllable service S^* for message bound $k \in \mathbb{N}$ on each message channel. Then, the set $Cov_{\varphi}(S^*)$ of all *deadlock-free covering situations* of service S^* is defined as:

$$Cov_{\varphi}(S^*) = \bigcup_{q \in Q_{mp_{df,k}(S^*)}} \xi_{\widehat{\Sigma}_{\varphi}(q)} \widehat{\mathcal{K}}_{\varphi}(q)$$

where $\xi_{\widehat{\Sigma}_{\varphi}(q)}(\widehat{\mathcal{K}}_{\varphi}(q))$ is an equivalence partition of the canonical stable situations $\widehat{\mathcal{K}}_{\varphi}(q)$ of S^* at state q of $mp_{df,k}(S^*)$ with respect to the set of canonical k-deadlock-free events $\widehat{\Sigma}_{\varphi}(q)$ at state q.

The equivalence relation on canonical stable situations of S^* with respect to canonical events partitions the canonical stable situations of S^* into sets of canonical stable situations. All stable situations in each set of the partition are equivalent in the sense that they force every deadlock-free controller of S^* to offer a canonical deadlock-free choice in a similar manner. Note, that each state q of a most permissive k-deadlock-free controller $mp_{df,k}(S^*)$ may be associated with multiple sets in a partition (cf. Example 6.3).

The union of all partitions at every state q of $mp_{df,k}(S^*)$ forms a set of deadlock-free covering situations of S^*.

We illustrate the calculation of the set $Cov_\varphi(S^*)$ of deadlock-free covering situations of service S^* with the following example.

Example 6.7. Consider a most permissive k-deadlock-free controller $mp_{df,k}({S_1}^*)$ of the complete canonical k-deadlock-free service $S_1^* = \mathcal{C}^2_{df,k}(S_1)$ from Figure 6.1. For $k = 1$ and each state of the controller $mp_{df,k}(S_1^*)$ of service S_1^*, the equivalence classes of the canonical stable situations of S_1^* with respect to the set of relevant canonical events are illustrated as follows:

$\xi_{\widehat{\Sigma}}(\widehat{\mathcal{K}}(L0)) = \{\ \{[2,[\ ]],\ [3,[\ ]],\ [11,[\ ]]\}\ \};$

$\xi_{\widehat{\Sigma}}(\widehat{\mathcal{K}}(L1)) = \{\ \{[3,[a]],\ [17,[\ ]]\}\ \};$

$\xi_{\widehat{\Sigma}}(\widehat{\mathcal{K}}(L2)) = \{\ \{[2,[b]],\ [30,[\ ]]\}\ \};$

$\xi_{\widehat{\Sigma}}(\widehat{\mathcal{K}}(L3)) = \{\ \{[43,[\ ]],\ [47,[\ ]],\ [50,[\ ]],\ [52,[\ ]]\}\ \}.$

Note that it is not necessarily the case where one set of canonical stable situations is associated with one subset in the partition. As in Example 6.3, we assume $\widehat{\mathcal{K}} = \{\ [q3,[a]],\ [q4,[a]],\ [q5,[b]]\ \}$ and $\widehat{E} = \{?a, ?b, \tau, \mathit{final}\}$. Thereby, $\widehat{\mathcal{K}}$ is partitioned into two sets of equivalent situations; these are, $\{[q_3,[a]],\ [q4,[a]]\}$ and $\{[q5,[b]]\}$. Then the equivalence class of $\widehat{\mathcal{K}}$ with respect to the set $\widehat{E}$ is $\xi_{\widehat{E}}(\widehat{\mathcal{K}}) = \{\ \{[q_3,[a]],\ [q4,[a]]\},\ \{[q5,[b]]\}\ \}$.

In case of S_1^*, the set $Cov_\varphi(S_1^*)$ of all *deadlock-free covering situations* of S_1^* from Figure 6.1 is illustrated by

$Cov_\varphi(S_1^*) = \{\ \{\ [2,[\ ]],\ [3,[\ ]],\ [11,[\ ]]\ \},\ \{\ [3,[a]],\ [17,[\ ]]\ \},\ \{\ [2,[b]],\ [30,[\ ]]\ \},\ \{\ [43,[\ ]],\ [47,[\ ]],\ [50,[\ ]],\ [52,[\ ]]\ \}\ \}.$ ◁

In the subsequent sections, we will show how to employ the set of deadlock-free covering situations to decide deadlock-free equivalence.

6.2.3. Matching Deadlock-free Covering Situations

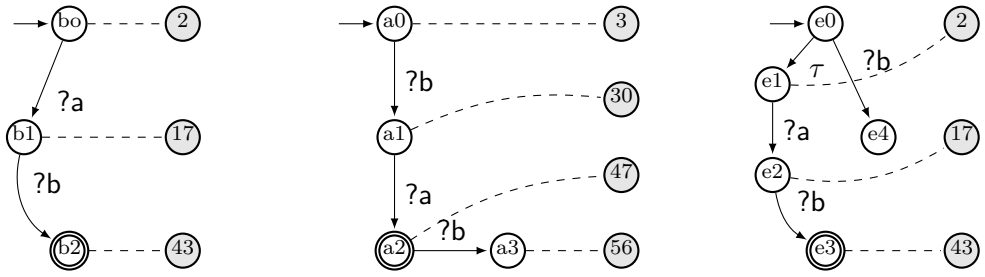
In this section, we define a matching relation between two services by comparing the specific subsets of their situations. This comparison takes into account the sets of deadlock-free covering situations of one service as described by Definition 6.6.

For this purpose, we first define the *stable failures cover relation* between two stable states as follows.

Definition 6.8 (Stable failures cover of stable states).
Let S and T be two interface equivalent service automata. Let q_S be a stable state in S and q_T be a stable state in T. Then, state q_T of T *covers* a stable failure of state q_S of S iff for each trace σ_S of S with $q_{0S} \stackrel{\sigma_S}{\Rightarrow}_S q_S$, there exists a trace σ_T of T with $q_{0T} \stackrel{\sigma_T}{\Rightarrow}_T q_T$ such that $\sigma_S = \sigma_T$ and $\mathit{Refuse}(q_S) = \mathit{Refuse}(q_T)$ holds.

Example 6.9. This example recalls service S_1 and S_2 from Figure 4.1 and a compact canonical k-deadlock-free service $S_1^* = \mathcal{C}^2_{df,k}(S_1)$ of S_1 from Figure 5.8 with message bound $k = 1$ for each message channel. The cover relations between states of S_2 and states of S_1^* are represented by dashed lines whereas states of S_1^* are represented by the shaded circle (the complete illustration of S_1^* is omitted due to its size).

Consider service S_2. State $b0$ is a stable state of S_2 with stable failure $[\epsilon, \{?b, \mathit{final}\}]$ and it covers the stable failure of state 2 of S_1^*. State $b1$ is a stable state of S_2 with stable failure $[?a, \{?a, \mathit{final}\}]$ and it covers stable failure of state 17 of S_1^*. State $b2$ is a stable state of S_2 with stable failure $[?a?b, \{?a, ?b\}]$ and it covers the stable failure of state 43 of S_1^*.



(a) S_2 (b) S_5 (c) S_9

Consider the two services S_5 and S_9 from Figure 6.4. The two services have an equivalent interface to service S_1. Services S_1 and S_5 are equivalent under k-deadlock freedom, however, services S_1 and S_9 are not. The cover relations between states of S_5 and states of S_1^* (also between states of S_9 and S_1^*) are represented by dashed lines whereas states of S_1^* are represented by the shaded circle.

Consider service S_5. State $a0$ is a stable state of S_5 with stable failure $[\epsilon, \{?a, \mathit{final}\}]$ and it covers state 3 of S_1^*. Also state $a1$ is a stable state of S_5 with stable failure $[?b, \{?b, \mathit{final}\}]$ and it covers state 30 of S_1^*. State $a2$ is a stable state of S_5 with stable failure $[?b?a, \{?a\}]$ and it covers state 47 of S_1^*. State $a3$ is a stable state of S_5 with stable failure $[?b?a?b, \{?a, ?b, final\}]$ and it covers state 56 of S_1^*.

Consider service S_9. State $e1$ is a stable state of S_9 with stable failure $[\epsilon, \{?b, \mathit{final}\}]$ and it covers state 2 of S_1^*. State $e2$ is a stable state of S_9 with stable failure $[?a, \{?a, \mathit{final}\}]$ and it covers state 17 of S_1^*. State $e3$ is a stable state of S_9 with stable failure $[?a?b, \{?a, ?b\}]$ and it covers state 43 of $\mathcal{C}^2_{df,k}(S_1)$. Though state $e4$ of S_9 does not cover stable failure of any state of S_1^*. ◁

We employ the stable failure cover of stable states to define the matching relation between service T and the set $Cov_\varphi(S^*)$ of all sets of deadlock-free covering situations of a complete canonical deadlock-free service of S.

Definition 6.10 (Matching deadlock-free covering situations).
Let S^* and T be two interface equivalent service automata. Let $\mu = \mathit{situations}(T)$ be a set of all situations of T with respect to the composition $T \oplus mp_{df,k}(S^*)$ of T and a most permissive k-deadlock-free controller $mp_{df,k}(S^*)$ of S^* for message bound $k \in \mathbb{N}$ for each message channel.

Then, service T *covers* the set $\mathit{Cov}_\varphi(S^*)$ of deadlock-free covering situations of service S iff for each $e_S \in \mathit{Cov}_\varphi(S^*)$ there exists $[q_S, \mathcal{M}_S] \in e_S$ and $[q_T, \mathcal{M}_T] \in \mu$ such that

1. q_T covers a stable failure of q_S, and
2. $\mathcal{M}_S = \mathcal{M}_T$.

The set of all service automata that cover the set $\mathit{Cov}_\varphi(S^*)$ of deadlock-free covering situations of S^* is denoted by $\mathit{f\text{-}Cover}(S^*)$.

Service T *covers* the set of deadlock-free covering situations of service S, whenever, for each subset $e_S \in \mathit{Cov}_\varphi(S^*)$ there are situations $[q_T, \mathcal{M}_T] \in e_S$ and $[q_T, \mathcal{M}_T]$ of T in which state q_T covers a stable failure state q_S and their state of message buffer are the same.

Example 6.11. Figure 6.2 shows the composition $S_2 \oplus mp_{df,k}(\mathcal{C}^2_{df,k}(S_1))$ of services S_2 from Figure 4.1 and a most permissive k-deadlock-free controller $mp_{df,k}(S_1^*)$ of $S_1^* = \mathcal{C}^2_{df,k}(S_1)$ from Figure 6.1 for message bound $k = 1$ on each message channel.

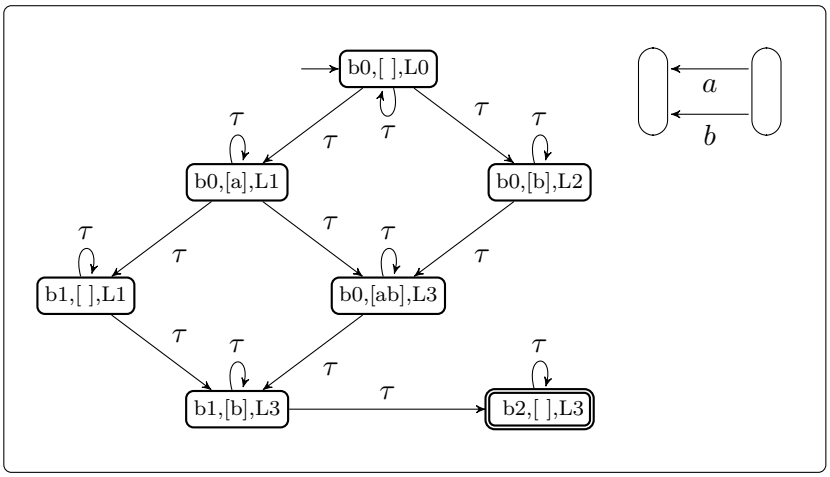


Figure 6.2.: The composition $S_2 \oplus mp_{df,k}(\mathcal{C}^2_{df,k}(S_1))$ of services S_2 and $mp_{df,k}(\mathcal{C}^2_{df,k}(S_1))$.

In this example, the set $\mu = \mathit{situations}(S_2)$ of all situations of S_2 with respect to the composition $S_2 \oplus mp_{df,k}(S_1^*)$ is illustrated by $\mu = \{\ [b0, [\,]],\ [b0, [a]],\ [b0, [b]],\ [b0, [ab]],\ [b1, [\,]],\ [b1, [b]],\ [b2, [\,]]\ \}$.

Recall $\mathit{Cov}_\varphi(S_1^*) = \{\ \{\ [2, [\,]],\ [3, [\,]],\ [11, [\,]]\ \},\ \{\ [3, [a]],\ [17, [\,]]\ \},\ \{\ [2, [b]],\ [30, [\,]]\ \},\ \{\ [43, [\,]],\ [47, [\,]],\ [50, [\,]],\ [52, [\,]]\ \}\ \}$ from Example 6.7 and the cover relations between

states of S_2 and states of S_1^* from Example 6.9. For each $e \in Cov_\varphi(S_1^*)$, we have

$e = \{[2,[\,]],[3,[\,]],[11,[\,]]\}$, state $b0$ covers stable failure of state 2 with $[b0,[\,]] \in \mu$,

$e = \{[3,[a]],[17,[\,]]\}$, state $b1$ covers stable failure of state 17 with $[b1,[\,]] \in \mu$,

$e = \{[2,[b]],[30,[\,]]\}$, state $b0$ covers stable failure of state 2 with $[b0,[b]] \in \mu$, and

$e = \{[43,[\,]],[47,[\,]],[50,[\,]],[52,[\,]]\}$, state $b2$ covers stable failure of state 43 with $[b2,[\,]] \in \mu$.

Therefore, S_2 covers the set $Cov_\varphi(S_1^*)$. ◁

In the following theorem, we prove one necessary condition of deadlock-free equivalence using the matching relation with the deadlock-free covering situations. That is, service T covers the set $Cov_\varphi(\mathcal{C}^2_{df,k}(S))$ of a complete canonical deadlock-free service $\mathcal{C}^2_{df,k}(S)$ of service S only if T is a substitute for S under k-deadlock-free equivalence.

Theorem 6.12 (Failures cover of two deadlock-freely equivalent services). For each and $k \in \mathbb{N}$ and each two interface equivalent services S and T:

$$T =_{df,k} S \quad \Rightarrow \quad T \text{ covers } Cov_\varphi(\mathcal{C}^2_{df,k}(S))$$

where $\mathcal{C}^2_{df,k}(S)$ is a complete canonical k-deadlock-free service of S.

Proof. Suppose $S^* = \mathcal{C}^2_{df,k}(S)$ and $T =_{df,k} S$. Because $S^* =_{df,k} S$ holds (Lemma 5.31), $T =_{df,k} S^*$ follows by transitivity.

Let $mp_{df,k}(T)$ and $mp_{df,k}(S^*)$ be most permissive k-deadlock-free controllers of T and S^* respectively. Because $T =_{df,k} S^*$, $mp_{df,k}(S^*)$ and $mp_{df,k}(T)$ are bisimilar (Corollary 3.56). Consider state q_{mT} in $mp_{df,k}(T)$ with $q_{0mT} \stackrel{\sigma_m}{\Rightarrow} q_{mT}$ and state q_m in $mp_{df,k}(S^*)$ with $q_{0m} \stackrel{\sigma_m}{\Rightarrow} q_m$, where q_{0mT} is an initial state of $mp_{df,k}(T)$ and q_{0m} is an initial state of $mp_{df,k}(S^*)$ respectively.

Let $\widehat{\varphi}(q_{mT})$ and $\widehat{\varphi}(q_m)$ be the set of canonical deadlock-free choices at q_{mT} and q_m respectively. Because $T =_{df,k} S^*$, $\widehat{\varphi}(q_m) = \widehat{\varphi}(q_{mT})$ (Corollary 3.56).

Because $mp_{df,k}(S^*)$ and $mp_{df,k}(T)$ are deterministic by construction, their sets of canonical deadlock-free events at q_m and at q_{mT} are equivalent, i. e.$\widehat{\Sigma}_\varphi(q_m) = \widehat{\Sigma}_\varphi(q_{mT})$. This means that the set $\widehat{\mathcal{K}}_\varphi(q_m)$ of all canonical deadlock-free situations at q_m and the set $\widehat{\mathcal{K}}_\varphi(q_{mT})$ of all canonical deadlock-free situations at q_{mT} preserves exactly the same set of canonical deadlock-free choices.

Consider a partition $\xi_{\widehat{\Sigma}_\varphi}(\widehat{\mathcal{K}}_\varphi(q_m))$ of equivalence situations that describes all equivalent situations of services in which one situation can replace others without effecting the canonical deadlock-free choices of $mp_{df,k}(S^*)$.

Consider the two following cases :

– $\widehat{\mathcal{K}}_\varphi(q_m) = \emptyset$: this means that either $\mathcal{K}(q_m) = \emptyset$ or $\mathit{stable}(\mathcal{K}(q_m)) = \emptyset$. The first case $\mathcal{K}_\varphi(q_m) = \emptyset$ means that state q_m is not reachable in the composition $S^* \oplus mp_{df,k}(S^*)$. In the latter case, assume $\mathcal{K}_\varphi(q_m) \neq \emptyset$. This means that every situation in $\mathcal{K}(q_m)$ is not stable. For both cases, $\xi_{\widehat{\Sigma}_\varphi(q_m)}(\widehat{\mathcal{K}}_\varphi(q_m)) = \emptyset$. There is no deadlock-free covering situation of S^* associated with states q_m.

– $\widehat{\mathcal{K}}_\varphi(q_m) \neq \emptyset$: Consider $e_S \in \mathit{Cov}_\varphi(S^*)$ and the partition $\xi_{\widehat{\Sigma}_\varphi(q_m)}(\widehat{\mathcal{K}}_\varphi(q_m))$ of canonical situations $\widehat{\mathcal{K}}_\varphi(q_m)$ with respect to canonical events $\widehat{\Sigma}_\varphi(q_m)$. This means that there exists a situation $[q_S, \mathcal{M}] \in e_S$ such that for each $[q'_S, \mathcal{M}'] \in e_S$ and for each $x \in \widehat{\Sigma}_\varphi(q_m)$ holds : $\mathit{activ}_k([q_S, \mathcal{M}], x) \Leftrightarrow \mathit{activ}_k([q'_S, \mathcal{M}'], x)$ by definition.

  As the situation $[q_S, \mathcal{M}]$ of S^* is a stable situation in $\mathcal{K}_\varphi(q_m)$, then for each $y \in \mathit{enable}(q_S)$ holds: $\mathit{step}_k([q_S, \mathcal{M}], y) = \mathit{false}$. This means that $\tau \notin \mathit{enable}(q_S)$ and q_S is a stable state of S^* follows. Therefore, there exists $[\sigma, X] \in \mathit{failures}(S^*)$ with $q_{0S} \stackrel{\sigma}{\Rightarrow} q_S$ and $X = \mathit{Refuse}(q_S)$.

  As $T =_{df,k} S$ implies $T \sqsubseteq_{df,k} S$, $\mathit{failures}(S^*) \supseteq \mathit{failures}(T)$ follows (Theorem 5.32).

  Suppose T is derived from S^* by removing each path $q_{0S} \stackrel{\sigma'}{\Rightarrow} q'_S$ that leads to state q'_S for every $[q'_S, \mathcal{M}'] \in e_S$ including the subsequent path that follows q'_S. This means, that there exists a subset $e_S \in \mathit{Cov}_\varphi(S^*)$ of deadlock-free situations of S^* such that for each situation $[q_T, \mathcal{M}_T]$ of T w. r. t.the composition $T \oplus mp_{df,k}(S^*)$ and for each situation $[q_S, \mathcal{M}_S]$ in e_S neither q_T can cover stable failure of q_S nor $\mathcal{M}_S = \mathcal{M}_T$ holds.

  As $T =_{df} S =_{df} S^*$, we have $q_{0mT} \stackrel{\sigma_m}{\Rightarrow} q_{mT}$ and $q_{0m} \stackrel{\sigma_m}{\Rightarrow} q_m$ and $\widehat{\varphi}(q_m) = \widehat{\varphi}(q_{mT})$. It follows that there exists $x \in \widehat{\Sigma}_\varphi(q_m)$ with $\mathit{activ}_k([q_S, \mathcal{M}], x) \Leftrightarrow \mathit{activ}_k([q'_S, \mathcal{M}'], x)$ but $x \notin \widehat{\Sigma}_\varphi(q_{mT})$ as all paths that results in each situation $[q_S, \mathcal{M}] \in e_S$ have been removed. This contradicts to the assumption $\widehat{\Sigma}_\varphi(q_m) = \widehat{\Sigma}_\varphi(q_{mT})$.

Thus, we conclude that T covers $\mathit{Cov}_\varphi(S^*)$. □

Theorem 6.12 suggests a necessary condition for deciding if two services are deadlock-freely equivalent.

Nevertheless, covering the set of all sets of deadlock-free covering situations is not sufficient to decide deadlock-free equivalence of a given service. This means, the reverse of Theorem 6.12 does not necessarily hold. As service T covers the set $\mathit{Cov}_\varphi(\mathcal{C}^2_{df,k}(S))$ of $\mathcal{C}^2_{df,k}(S)$ of service S only means that T covers the states that preserves all canonical deadlock-free choices of all controllers of S, but it does not guarantee that the behavior of T will not deadlock with every deadlock-free controller of service S.

We illustrate this circumstance with the following example.

Example 6.13. Consider services S_9 from Figure 6.4 and Example 6.9 with message bound $k = 1$ on each message channel. Service S_9 is not equivalent to service S_1 from Figure 4.1 under k-deadlock-freedom. Though S_9 covers the set $\mathit{Cov}_\varphi(S_1^*)$ of $S_1^* = \mathcal{C}^2_{df,k}(S_1)$ from Example 6.7.

Figure 6.3 shows the composition $S_9 \oplus mp_{df,k}(\mathcal{C}^2_{df,k}(S_1))$ of service S_9 and a most permissive k-deadlock-free controller $mp_{df,k}(S_1^*)$ of S_1^* from Figure 6.1. The set $\mu =$

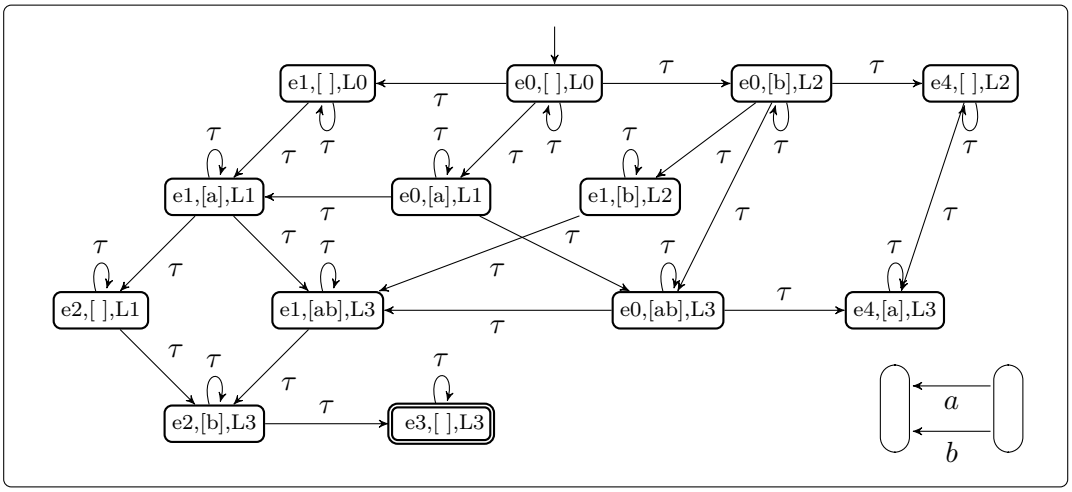


Figure 6.3.: The composition $S_9 \oplus mp_{df,k}(\mathcal{C}^2_{df,k}(S_1))$ of services S_9 and $mp_{df,k}(\mathcal{C}^2_{df,k}(S_1))$.

situations(S_9) of all situations of S_9 with respect to the composition $S_9 \oplus mp_{df,k}(S_1^*)$ is illustrated by $\mu = \{\ [e0,[\ ]],\ [e0,[a]],\ [e0,[b]],\ [e0,[ab]],\ [e1,[\ ]],\ [e1,[a]],\ [e1,[b]],\ [e1,[ab]],\ [e2,[\ ]],\ [e2,[b]],\ [e3,[\ ]],\ [e4,[\ ]],\ [e4,[a]]\ \}$.

Recall $Cov_\varphi(S_1^*) = \{\ \{\ [2,[\ ]],\ [3,[\ ]],\ [11,[\ ]]\ \},\ \{\ [3,[a]],\ [17,[\ ]]\ \},\ \{\ [2,[b]],\ [30,[\ ]]\ \},\ \{\ [43,[\ ]],\ [47,[\ ]],\ [50,[\ ]],\ [52,[\ ]]\ \}\ \}$ from Example 6.7 and the cover relations between states of S_9 and states of S_1^* from Example 6.9. For each $e \in Cov_\varphi(S_1^*)$, we have

$e = \{[2,[\ ]],[3,[\ ]],[11,[\ ]]\}$, state $e1$ covers stable failure of state 2 with $[e1,[\ ]] \in \mu$,

$e = \{[3,[a]],[17,[\ ]]\}$, state $e2$ covers stable failure of state 17 with $[e2,[\ ]] \in \mu$,

$e = \{[2,[b]],[30,[\ ]]\}$, state $e1$ covers stable failure of state 2 with $[e1,[b]] \in \mu$, and

$e = \{[43,[\ ]],[47,[\ ]],[50,[\ ]],[52,[\ ]]\}$, state $e3$ covers stable failure of state 43 with $[e3,[\ ]] \in \mu$.

Therefore, S_9 covers the set $Cov_\varphi(S_1^*)$. ◁

In the next section, we show necessary and sufficient condition for deciding deadlock freedom equivalence of a given service S. That is, service S and T are deadlock-freely equivalent whenever T refines $\mathcal{C}^2_{df,k}(S)$ under stable failures and T covers the set $Cov_\varphi(S^*)$ of all sets of deadlock-free covering situations of $\mathcal{C}^2_{df,k}(S)$.

6.2.4. Deadlock-free Equivalence Guidelines

In this section, we prove necessary and sufficient conditions for deciding deadlock freedom equivalence of a given service. In the following theorem, we show that refining a canonical deadlock-free service under stable failures and covering the set of deadlock-free covering situations of a canonical deadlock-free service together is necessary and sufficient to decide deadlock-free equivalence of a given service.

Theorem 6.14 (Characterizing deadlock-freely equivalent services).
For each message bound $k \in \mathbb{N}$ and each two k-deadlock-freely controllable service S and T:

$$df_k Equiv(S) \quad = \quad f\text{-}refine(\mathcal{C}^2_{df,k}(S)) \ \cap \ f\text{-}Cover(\mathcal{C}^2_{df,k}(S))$$

where $\mathcal{C}^2_{df,k}(S)$ is a complete canonical k-deadlock-free service of S.

Proof. Let $S^* = \mathcal{C}^2_{df,k}(S)$. We prove this theorem in two directions.

$\Rightarrow$: Suppose $T =_{df,k} S$. This means that $T \in f\text{-}refine(S^*)$ (Theorem 5.32) and $T \in f\text{-}Cover(Cov_\varphi(S^*))$ (Theorem 6.12).

$\Leftarrow$: Suppose $S^* \sqsubseteq_{SF} T$ and T covers $Cov_\varphi(S^*)$. Because $S^* \sqsubseteq_{SF} T$ holds by assumption, $T \sqsubseteq_{df,k} S =_{df,k} S^*$ follows (Theorem 5.32 and Lemma 5.31).

Let $mp_{df,k}(T)$ and $mp_{df,k}(S^*)$ be most-permissive deadlock-free controllers of T and S^* respectively. We will show that (1) $mp_{df,k}(S^*)$ and $mp_{df,k}(T)$ are bisimilar and (2) for each pair $[q_{mT}, q_{mS^*}]$ of bisimilar states holds: $\varphi(q_{mT}) = \varphi(q_{mS^*})$.

1. Because $T \sqsubseteq_{df,k} S^*$, then there exists a strong simulation relation ϱ such that $mp_{df,k}(T)$ simulates $mp_{df,k}(S^*)$ with ϱ (Corollary 3.45).

   By construction, $mp_{df,k}(S^*)$ and $mp_{df,k}(T)$ are derived from the initial situations $\{[q_{0S^*}, [\,]]\}$ of S^* and $\{[q_{0T}, [\,]]\}$ of T using the closure and event operations on knowledge of $mp_{df,k}(S^*)$ and $mp_{df,k}(T)$ respectively.

   Let $[q_{0mT}, q_{0mS^*}] \in \varrho^*$ be a binary relation between states of $mp_{df,k}(T)$ and $mp_{df,k}(S^*)$.

   Consider $[q_{mT}, q_{mS^*}] \in \varrho^*$ with $[q_{S^*}, \mathcal{M}_{S^*}] \in stable(\mathcal{K}(q_{mS^*}))$ and $[q_{S^*}, \mathcal{M}_{S^*}] \in e \in Cov_\varphi(S^*)$ with $q_{0S^*} \stackrel{\sigma_*}{\Rightarrow} q_{S^*}$ and $Refuse(q_{S^*}) = X^*$.

   By assumption, $failures(S^*) \supseteq failures(T)$ holds. Then for each $[\sigma_T, X_T] \in failures(T)$, there is $[\sigma_{S^*}, X_{S^*}] \in failures(S^*)$ with $\sigma_{S^*} = \sigma_T$ and $X_{S^*} = X_T$.

   By assumption, T covers the set $Cov_\varphi(S^*)$. Then there exists a situation $[q_T, \mathcal{M}_T]$ in T such that (1) q_T covers a stable failure of q_{S^*} and (2) $\mathcal{M}_S = \mathcal{M}_T$. This means that if there is a stable failure that is described by S^* but not by T, then such a failure does not affect the canonical stable situations at q_{mS^*} and it does not affect the canonical choices at state q_{mS^*}. This is because there is a situation of T that preserves each subset of deadlock-free covering situations of S^*.

   For each event $m \in \Sigma$, the successor state of state q_{mT} is uniquely determined by $\mathcal{K}'(q_{mT}) = closure(event(\mathcal{K}(q_{mT})), m)$ of $mp_{df,k}(T)$ by construction. By assumption, S^* and T have the same set of input I and output O message channels, and the set of all situations of both S^* and T are bounded by k. Then, there exists also the successor state of state q_{mS^*} that is uniquely determined by $\mathcal{K}'(q_{mS^*}) = closure(event(\mathcal{K}(q_{mS^*})), m)$ of $mp_{df,k}(S^*)$ by construction.

As $mp_{df,k}(S^*)$ and $mp_{df,k}(T)$ are interface equivalent and deterministic, the relation ϱ^* is a strong simulation relation such that $mp_{df,k}(S^*)$ strongly simulates $mp_{df,k}(T)$ with ϱ^*. As $mp_{df,k}(T)$ also strongly simulates $mp_{df,k}(S^*)$ with ϱ and $mp_{df,k}(S^*)$ and $mp_{df,k}(T)$ are deterministic, this means that $mp_{df,k}(S^*)$ and $mp_{df,k}(T)$ are bisimilar.

2. Consider a pair $[q_{mT}, q_{mS^*}]$ of bisimilar states of $mp_{df,k}(S^*)$ and $mp_{df,k}(T)$. It follows from the bisimulation relation that $enable(q_{mT}) = enable(q_{mS^*})$.

   Consider $\mathcal{K}(q_{mS^*})$ and $\mathcal{K}(q_{mT})$. By construction, $mp_{df,k}(S^*)$ is a k-deadlock-free controller of S^* and $mp_{df,k}(T)$ is a k-deadlock-free controller of T. Because $enable(q_{mT}) = enable(q_{mS^*})$ also holds, it follows that the set of legally updated events at q_{mS^*} and that at q_{mT} are the same set, i. e., $\mathcal{L}_k(\mathbb{E}_f, \mathcal{K}(q_{mS^*})) = \mathcal{L}_k(\mathbb{E}_f, \mathcal{K}(q_{mT}))$.

   Consider $\mathcal{K}(q_{mS^*})$ and the following cases:

   – $\mathcal{K}(q_{mS^*}) = \emptyset$: this means that q_{mS^*} is a state that is never reached in the composition of S^* with any of its controllers. As $[q_{mT}, q_{mS^*}]$ is a pair of bisimilar states, this means $\mathcal{K}(q_{mT}) = \emptyset$ also holds and q_{mT} is also not reachable in the composition of T with any of its controllers. As every failure of T is also a failure of S^* by assumption, there is no failure of T that is not described by S^*. That is, $\varphi(q_{mT}) = \varphi(q_{mS^*})$ holds.

   – $\mathcal{K}(q_{mS^*}) \neq \emptyset$ and $stable(\mathcal{K}(q_{mS^*})) = \emptyset$: this means that every corresponding state of q_{mS^*} in S^* can make a move without help from a controller of S^*. Similarly to the case $\mathcal{K}(q_{mS^*}) = \emptyset$, none of situations in $\mathcal{K}(q_{mT})$ contributes to a deadlock-free covering situation of S^*. Hence, $\varphi(q_{mT}) = \varphi(q_{mS^*})$ follows.

   – $stable(\mathcal{K}(q_{mS^*})) \neq \emptyset$: Because $T \sqsubseteq_{df,k} S^*$ holds, then $[q_{mS^*}, q_{mT}] \in \varrho$ is a strong simulation relation and $\varphi(q_{mS^*}) \subseteq \varphi(q_{mT})$ holds (Lemma 3.52)

     Let $ch \in \varphi(q_{mT})$, we will show that $ch \in \varphi(q_{mS^*})$ holds.

     Let $[q_{S^*}, \mathcal{M}_{S^*}] \in e_S \in Cov_\varphi(S^*)$. Consider $q_{0S^*} \stackrel{\sigma}{\Rightarrow} q_{S^*}$ with $X = Refuse(q_{S^*})$.

     By assumption, T covers the set $Cov_\varphi(S^*)$. Then there exists situation $[q_T, \mathcal{M}_T]$ in T such that (1) q_T covers stable failure of q_{S^*} and (2) $\mathcal{M}_S = \mathcal{M}_T$.

     Consider a pair $[q_{mT}, q_{mS^*}]$ of bisimilar states with $[q_T, \mathcal{M}_T] \in stable(\mathcal{K}(q_{mT}))$ and $[q_{S^*}, \mathcal{M}_{S^*}] \in stable(\mathcal{K}(q_{mS^*}))$.

     As $ch \in \varphi(q_{mT})$, there exists $m \in ch$ such that $activ_k([q_T, \mathcal{M}], m) = true$. As $\mathcal{L}_k(\mathbb{E}_f, \mathcal{K}(q_{mS^*})) = \mathcal{L}_k(\mathbb{E}_f, \mathcal{K}(q_{mT}))$ and $Refuse(q_T) = Refuse(q_{S^*})$ holds, then $activ_k([q_{S^*}, \mathcal{M}^*], m) = true$ also holds. This means that $ch \in \varphi(q_{mS^*})$ follows.

     Therefore, $\varphi(q_{mS^*}) \supseteq \varphi(q_{mT})$ holds.

   As $mp_{df,k}(S^*)$ and $mp_{df,k}(T)$ are bisimilar and for each pair $[q_{mT}, q_{mS^*}]$ of bisimilar states $\varphi(q_{mT}) = \varphi(q_{mS^*})$ holds, $T =_{df,k} S$ follows (Lemma 3.52).

Thus, the theorem holds. □

The value of Theorem 6.14 is for characterizing all services that are deadlock-freely equivalent to a given service S. Two services S and T are deadlock-freely equivalent whenever (1) T *refines* the canonical deadlock-free service $\mathcal{C}^2_{df,k}(S)$ of S under stable failures and (2) T *covers* the set $Cov_\varphi(\mathcal{C}^2_{df,k}(S))$ of all sets of deadlock-free covering situations of the canonical deadlock-free service $\mathcal{C}^2_{df,k}(S)$ of S.

We present a *deadlock-free equivalence guideline* as a finite representation of all k-deadlock-freely equivalent services of a given service S as follows.

Definition 6.15 (Deadlock-free equivalence guideline).
Let $k \in \mathbb{N}$ a message bound for each message channel and $\mathcal{C}^2_{df,k}(S)$ be the complete canonical k-deadlock-free service of a k-deadlock-freely controllable service automaton S. Let $Cov_\varphi(\mathcal{C}^2_{df,k}(S))$ be the set of all sets of deadlock-free cover states of $\mathcal{C}^2_{df,k}(S)$. Then, a *deadlock-free equivalence guideline* of service S is denoted by the pair $[\mathcal{C}^2_{df,k}(S), Cov_\varphi(\mathcal{C}^2_{df,k}(S))]$.

A service T *matches* with $[\mathcal{C}^2_{df,k}(S),\ Cov_\varphi(\mathcal{C}^2_{df,k}(S))]$ of service S iff T refines $\mathcal{C}^2_{df,k}(S)$ under stable failures and T covers $Cov_\varphi(\mathcal{C}^2_{df,k}(S))$.

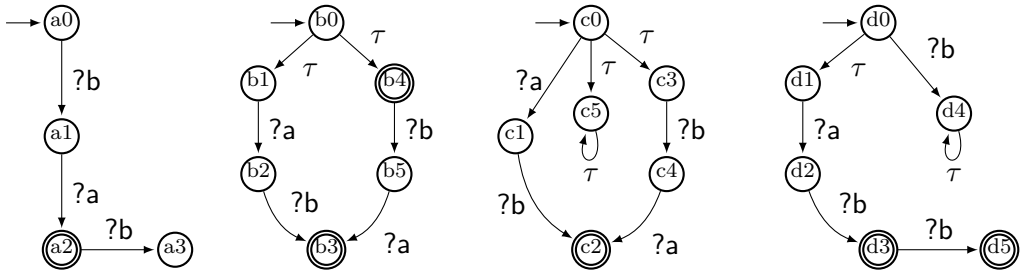
For characterizing the set of all deadlock-freely equivalent substitutes for a given service S, it is not possible to replace the complete canonical deadlock-free substitute $\mathcal{C}^2_{df,k}(S)$ for S in an equivalence guideline of S by a compact canonical deadlock-free substitute $\widehat{\mathcal{C}}^2_{df,k}(S)$ for S. This is because, by construction, the compact canonical deadlock-free substitute $\widehat{\mathcal{C}}^2_{df,k}(S)$ for S does not encode within its structure all possible deadlock-free choices of controllers, but only essential choices called canonical deadlock-free choices. Nevertheless, we can optimize the construction procedure of the service $\mathcal{C}^2_{df,k}(S)$ by employing the compact canonical deadlock-free controller $\widehat{\mathcal{C}}_{df,k}(S)$ of S instead of the complete canonical deadlock-free controller $\mathcal{C}_{df,k}(S)$ of S. We discuss more on this issue in Chapter 7.

We illustrate how deadlock-free equivalence guidelines characterize the set of deadlock-freely equivalent services with the two following examples.

Example 6.16. Consider services S_1, S_2, S_3, and S_4 from Figure 4.1. Each has the same interface, i. e., $I = \{a, b\}$ and $O = \{\ \}$. For service S_1, a deadlock-free equivalence guideline of S_1 is illustrated by $[\mathcal{C}^2_{df,k}(S_1), Cov_\varphi(\mathcal{C}^2_{df,k}(S_1))]$ from Example 6.7 for message bound $k = 1$ on each message channel.

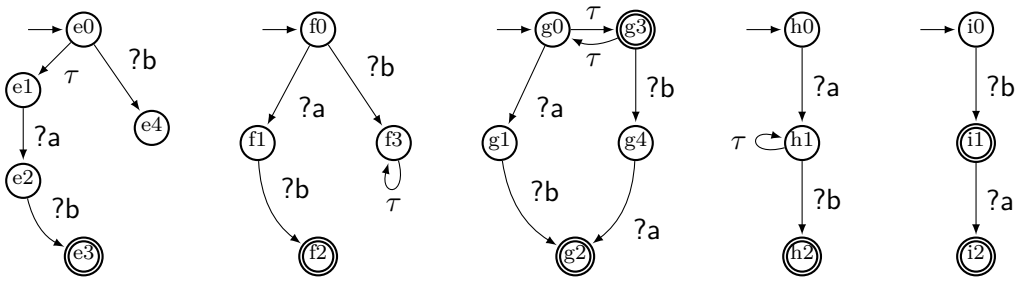
Though both services S_3 and S_4 refine the complete canonical k-deadlock-free service $\mathcal{C}^2_{df,k}(S_1)$ of S under stable failures, they are not equivalent to service S_1 and S_2 under k-deadlock freedom. This is because S_3 and S_4 do not cover $Cov_\varphi(\mathcal{C}^2_{df,k}(S_1))$. There exists $e_1 \in Cov_\varphi(\mathcal{C}^2_{df,k}(S_1))$ with $e_1 = \{\ [2, [\ ]], [3, [\ ]], [11, [\ ]]\ \}$ where there is no $q \in e_1$ and no q_{S_3} of S_3 such that q_{S_3} can cover stable failures of state $q \in e_1$. Similar to service S_3, service S_4 cannot cover $e_1 \in Cov_\varphi(\mathcal{C}^2_{df,k}(S_1))$. ◁

Positive Instances characterized by an equivalence guideline of S_1 *for* $k = 1$



(a) S_5 (b) S_6 (c) S_7 (d) S_8

Negative Instances characterized by an equivalence guideline of S_1 *for* $k = 1$



(e) S_9 (f) S_{10} (g) S_{11} (h) S_{12} (i) S_{13}

Figure 6.4.: Nine services: S_5, S_6, S_7, S_8, S_9, S_{10}, S_{11}, S_{12}, and S_{13}; each with $I = \{a, b\}$ and $O = \{\,\}$, in comparison the deadlock-free equivalence guidelines $[\mathcal{C}^2_{df,k}(S_1), Cov_\varphi(\mathcal{C}^2_{df,k}(S_1))]$ of service S_1 from Figure 4.1 for message bound $k = 1$ on each message channel.

Example 6.17. Consider services S_5, S_6, S_7, S_8, S_9, S_{10}, S_{11}, S_{12}, and S_{13} in Figure 6.4. Each has $I = \{a, b\}$ and $O = \{\,\}$, the same interface as service S_1 from Figure 4.1.

Consider the deadlock-free equivalence guideline $[\mathcal{C}^2_{df,k}(S_1), Cov_\varphi(\mathcal{C}^2_{df,k}(S_1))]$ of service S_1 from Example 6.7 for message bound $k = 1$ on each message channel. The four services S_5, S_6, S_7, and S_8 are equivalent to service S_1 and are characterized by the equivalence guideline of S_1. The five services S_9, S_{10}, S_{11}, S_{12}, and S_{13} are not equivalent to service S_1, and therefore, are not characterized by the equivalence guideline of S_1.

We see that each service in Figure 6.4 except service S_9 refines the service $\mathcal{C}^2_{df,k}(S_1)$ under stable failures. This means that every service except for service S_9 is a substitute for service S_1 under deadlock freedom as asserted by Theorem 5.32.

Though both services S_8 and S_{10} can diverge after receiving the first message b at the initial state, S_8 and S_{10} are not in the same deadlock-free equivalence class. This is because the initial state $d0$ of service S_8 is not a stable state, and S_8 can perform an

internal τ event and waits at state $d1$ for its controller to offer a choice. At state $d1$, it is no longer possible for service S_8 to diverge. However, this is not the case for service S_{10}, as the initial state $f0$ of S_{10} is a stable state. This means that service S_{10} waits at its initial state $d1$ for its controller to offer a choice. It is still possible for S_{10} to diverge after consuming message b.

We see that service S_8 matches with the equivalence guidelines of S_1, this means, S_8 is equivalent to S_1 under k-deadlock freedom. Nevertheless, service S_{10} dose not match with the equivalence guideline of S_1, because S_{10} cannot cover failures of any state from $e = \{\ [2, [\ ]],\ [3, [\ ]],\ [11, [\ ]]\ \} \in Cov_{\varphi}(\mathcal{C}^2_{df,k}(S_1))$. ◁

6.3. Characterization of all Responsively Equivalent Substitutes

Two services are responsively equivalent whenever they have the same set of responsive controllers. Nevertheless, one service possibly has completely different order of events than another service in its own responsive equivalence class. In Section 5.2.2 Chapter 5, we introduce a complete canonical responsive service of a given service as a service that is equivalent to the given service under deadlock freedom. We show that a complete canonical responsive service is structurally more liberal than any other service in the same responsiveness equivalence class in the sense that it encodes all possible sequence and choice of events within its own structure.

In this section, we study a complete canonical substitute of a given service and investigate how it influences all its responsive controllers. For this, we first define in Section 6.3.1 canonical wait situations as representative of situations that possibly influence its responsive controllers. In Section 6.3.2, we employ the equivalence relation on situations to partition the relevant canonical wait situations and calculate sets of *covering situations* of the complete canonical responsive substitute that associates with the calculated partition. In Section 6.3.3, we define a matching relation of covering situations and propose one necessary condition for deciding deadlock freedom based on the matching relation. Finally, we present in Section 6.3.4 a responsive equivalence guidelines for characterizing the set of all responsively equivalent services.

Several underlying concepts and techniques for characterizing all responsively equivalent services are similar to the characterization of all deadlock-freely equivalent services. The definitions, lemmas, and theorems defined in this section follow analogously from Section 6.2, though subtle differences lie in their technical details.

6.3.1. Canonical Wait Situations

In this section, we study a situation of the given service that influences how its responsive controllers behave. We first recall from Section 3.3.3 a *wait situation* is a situation in which service *cannot perform an communicating event* without any help from its interacting partner. To interact with the service in a responsive manner, a controller must offer a choice that can resolve the wait situation. Different controller may offer different choices to resolve the same situation. Nevertheless, there are some wait situations which

are more influential to its controllers than other wait situations, as every controller *must* provide the same choice that can resolve such a situation in a responsive manner.

For this purpose, we define a *canonical wait situation* as a wait situation of a given service that can influence exactly the responsive choices that every controller of the service must provide.

Definition 6.18 (Canonical wait situations).
Let $\mathcal{K}(q)$ be the knowledge at state q of service P about its interface compatible service S. Let $\psi(q)$ be the set of responsive choices, $\psi^*(q)$ be the set of valid responsive choices, and $\widehat{\psi^*}(q)$ be the set of canonical valid responsive choices at state q. Let E^* and $\widehat{E^*}$ be the set of all events that are described by $\psi(q)$ and $\widehat{\psi^*}(q)$ respectively. Let $\mathcal{K}^* = \mathit{wait}(\mathcal{K}(q))$ be a set of all wait situations at state q with $[q, \mathcal{M}] \in \mathcal{K}^*$.

Then, the wait situation $[q, \mathcal{M}]$ is a *canonical wait situation* at state q if for each event $e \in E^*$ and $e \notin \widehat{E^*}$ hods: $\mathit{activ}_k([q, \mathcal{M}], e) = \mathit{false}$.

The set of all canonical wait situations at state q of P is denoted by $\widehat{\mathcal{K}}_{\psi^*}(q)$.

Recall from Chapter 3. A valid responsive choice $\psi^*(q)$ at state q of a given service P is a responsive choice in which an event of the choice either is enabled at state q or is a *final* event (Definition 3.60). A canonical valid responsive choice $\widehat{\psi^*}(q)$ at state q of service P is a valid responsive choice that the service must provide in order to resolve the situations of its interacting partner (e. g., service S) in a responsive manner (Definition 3.27).

Given a set $\mathcal{K}^* = \mathit{wait}(\mathcal{K}(q))$ of wait situations of S at state q of P, a *canonical wait situation* of $\mathcal{K}$ is a situation of S that can only be updated by a canonical valid responsive event of P, not by any other event of P that is described by a valid responsive choice. The calculation of canonical wait situations does not take into account an event that is described by an invalid responsive choice.

The following example shows the canonical wait situations of a complete canonical k-deadlock-free service of a given service when interacting with its most permissive k-responsive controller.

Example 6.19. Figure 6.5 shows a most permissive k-responsive controller $mp_{rp,k}(A_8)$ of service ${A_8}^*$ from Figure 2.8 with its responsive choices ψ^* and a complete canonical k-responsive controller $\mathcal{C}_{rp,k}(A_8)$ of A_8 that is constructed from $mp_{rp,k}(A_8)^{\psi^*}$. Figure 6.6 shows a most permissive k-responsive controller $mp_{rp,k}(\mathcal{C}_{rp,k}(A_8))^{\psi^*}$ of the controller $\mathcal{C}_{rp,k}(A_8)$ with its responsive choices ψ^* and a complete canonical k-responsive service $A_8^* = \mathcal{C}^2_{rp,k}(A_8)$ of A_8 that is constructed from $mp_{rp,k}(\mathcal{C}_{rp,k}(A_8))^{\psi^*}$.

Consider a most permissive k-responsive controller $mp_{rp,k}({A_8}^*)$ of the complete canonical k-responsive service A_8^* from Figure 6.7 for message bound $k = 1$ on each message channel. The controller $mp_{rp,k}(A_8^*)$ consists of five states; q_1, q_2, q_3, q_4, and q_5. Each state q of the controller $mp_{df,k}(A_8^*)$ represents its knowledge of all relevant situations of A_8^* that are associated with state q in the composition $A_8^* \oplus mp_{df,k}(A_8^*)$ of A_8^* and $mp_{df,k}(A_8^*)$. The underlined situations denote the wait situations of A_8^*.

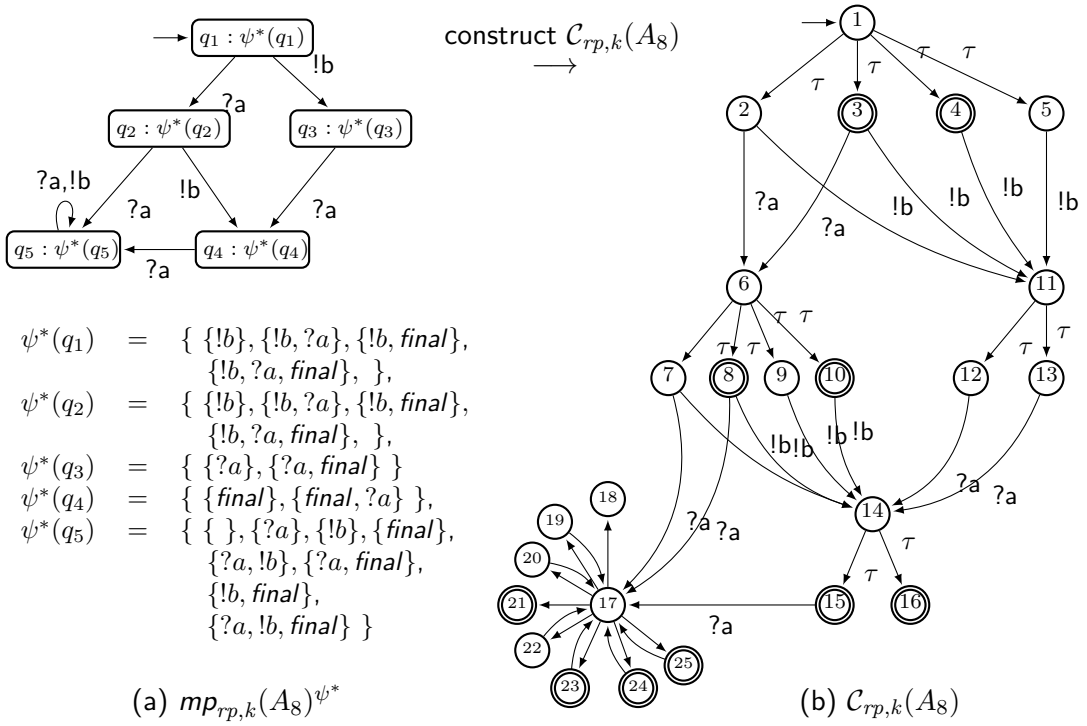


Figure 6.5.: A k-responsive operating guideline $mp_{rp,k}(A_8)^{\psi^*}$ of A_8 (from Figure 2.8) and a complete canonical k-responsive controller; $\mathcal{C}_{rp,k}(A_8)$ of A_8, both with $I = \{a\}$ and $O = \{b\}$ for message bound $k = 1$.

The two controllers $mp_{rp,k}(A_8)$ and $mp_{rp,k}(A_8^*)$ are bisimilar. This is because A_8 and A_8^* are equivalent under k-responsiveness (asserted by Lemma 5.35). The set of canonical responsive choices of the two controllers $mp_{rp,k}(A_8)$ and $mp_{rp,k}(A_8^*)$ at each pair of bisimilar states are also the same set (asserted by Corollary 3.74). Though the situations of A_8^* associated with the controller $mp_{rp,k}(A_8^*)$ are different than situations of A_8 associated with the controller $mp_{rp,k}(A_8)$ (not shown here). By construction, the structure of A_8^* encodes all possible sequence and choice of events within its own structure.

For state q_1 of $mp_{rp,k}(A_8^*)$, the set $\widehat{\varphi}(q_1)$ of all canonical responsive choices is $\widehat{\varphi}(q_1) = \{\{!b\}\}$ and the set $\widehat{\Sigma}_\varphi(q_1)$ of all events described by $\widehat{\varphi}(q_1)$ is $\widehat{\Sigma}_\varphi(q_1) = \{!b\}$. To calculate $\widehat{\mathcal{K}}(q_1)$, we consider the activation of each event for wait situations at q_1 in the following table.

$[q, \mathcal{M}] \in wait(\mathcal{K}(q_1))$	[7, [ ]]	[8, [ ]]	[10, [a]]	[11, [a]]
$activ_k([q, \mathcal{M}], ?a)$	*false*	*false*	*true*	*true*
$activ_k([q, \mathcal{M}], !b)$	*true*	*true*	*true*	*true*
$activ_k([q, \mathcal{M}], final)$	*false*	*true*	*false*	*false*
$enable(q)$	$\{?b\}$	$\{?b, final\}$	$\{?b\}$	$\{?b, final\}$

The three wait situations $[8, [\,]]$, $[10, [a]]$, $[11, [a]]$ are not canonical situations in $\widehat{\varphi}(q_1)$. This

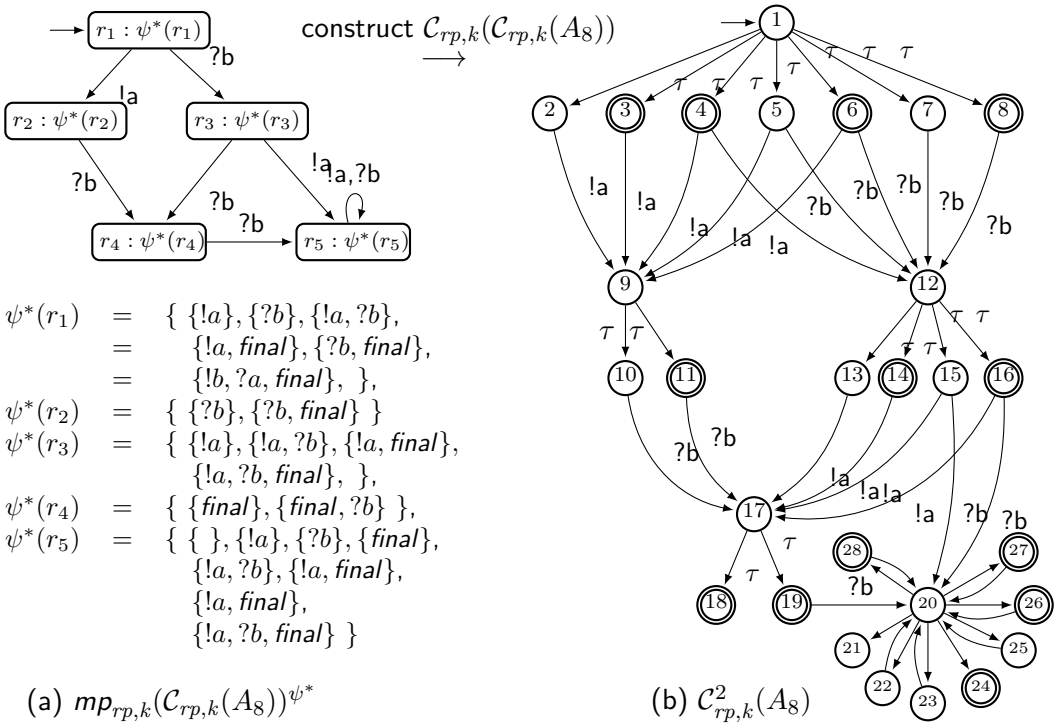


Figure 6.6.: A k-responsive operating guideline $mp_{rp,k}(\mathcal{C}_{rp,k}(A_8))^{\psi^*}$ of canonical k-responsive controller $\mathcal{C}_{rp,k}(A_8)$ (from Figure 6.5) and a complete canonical k-responsive substitute $\mathcal{C}^2_{rp,k}(A_8)$ for A_8, both with $I = \{b\}$ and $O = \{a\}$ for message bound $k = 1$.

is because $activ_k([8, [\,]], final) = true$ and $activ_k([10, [a]], ?a) = activ_k([11, [a]], ?a) = true$, but $?a$ and *final* are not canonical events at state q_1 (*i. e.*, $?a, final \notin \widehat{\Sigma}_\varphi(q_1)$). Therefore, $[7, [\,]]$ is the only canonical wait situation at state q_1.

The canonical situations of (A_8^*) from Figure 6.7 are illustrated as follows:

$\widehat{\Sigma}_{\psi^*}(q_1) = \{!b\}$ and $\widehat{\mathcal{K}}(q_1) = \{[7, [\,]]\}$,

$\widehat{\Sigma}_{\psi^*}(q_2) = \{!b\}$ and $\widehat{\mathcal{K}}(q_2) = \{[10, [\,]]\}$,

$\widehat{\Sigma}_{\psi^*}(q_3) = \{?a\}$ and $\widehat{\mathcal{K}}(q_3) = \{[18, [a]], [19, [a]]\}$,

$\widehat{\Sigma}_{\psi^*}(q_4) = \{final\}$ and $\widehat{\mathcal{K}}(q_4) = \{[18, [\,]], [19, [\,]]\}$, and

$\widehat{\Sigma}_{\psi^*}(q_5) = \{?a, !b, final\}$ and $\widehat{\mathcal{K}}(q_5) = \{\}$.

In this example, the calculation of $\widehat{\mathcal{K}}(q_2)$, $\widehat{\mathcal{K}}(q_3)$, $\widehat{\mathcal{K}}(q_4)$, and $\widehat{\mathcal{K}}(q_5)$ are omitted as it is almost similar to the calculation of $\widehat{\mathcal{K}}(q_1)$. $\lhd$

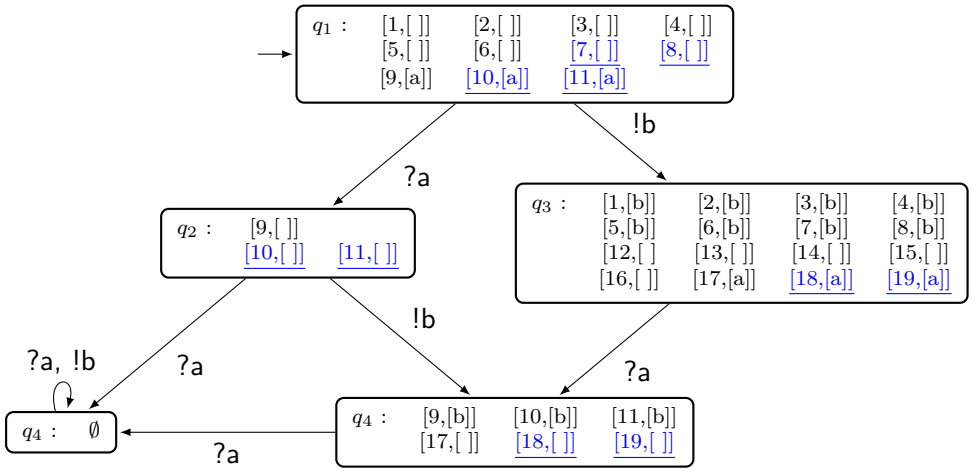


Figure 6.7.: A most permissive k-responsive controller $mp_{rp,k}(\mathcal{C}^2_{rp,k}(A_8))$ of a canonical k-deadlock-free substitute $\mathcal{C}^2_{rp,k}(A_8)$ from Figure 6.6 for $k = 1$. Each state of the controller contains knowledge of $\mathcal{C}^2_{rp,k}(A_8)$.

6.3.2. Responsive Covering Situations

In the previous section, we introduce the canonical wait situations as the situations which can influence exactly the canonical responsive choices; that are the choices in which every responsive controller the given service must provide. Inherently not every canonical wait situation has an influence on the canonical choices in a similar way. With respect to a set of events described by the canonical responsive choices, two equivalent canonical wait situations intuitively means each situation can act on behalf of another without affecting a canonical responsive choice of a responsive controller.

In this section, we employ the equivalence relation on situations as defined in Section 6.1 to calculate the set of *responsive covering situations* of a given service S^*. For each state q of a most permissive responsive controller of S^*, we employ the equivalence relation to partition the relevant canonical wait situations of S^* with respect to the set of events described by canonical valid responsive choices at q. Then we calculate the set of *responsive covering situations* of S^* by performing a union operation of all subsets of all partitions at all states of a most permissive responsive controller of S^*.

We define the set $Cov_\psi(S^*)$ of *responsive covering situations* of service S^* as follows.

Definition 6.20 (Responsive covering situations).
Let $mp_{rp,k}(S^*)$ be a most permissive k-responsive controller of a k-responsively controllable service S^* for message bound $k \in \mathbb{N}$ on each message channel. Then, the set $Cov_\psi(S^*)$ of all *responsive covering situations* of service S^* is defined as:

$$Cov_\psi(S^*) = \bigcup_{q \in Q_{mp_{rp,k}(S^*)}} \xi_{\widehat{\Sigma}_\psi(q)} \widehat{\mathcal{K}}_\psi(q)$$

where $\xi_{\widehat{\Sigma}_\psi(q)}(\widehat{\mathcal{K}}_\psi(q))$ is an equivalence partition of canonical wait situations $\widehat{\mathcal{K}}_\psi(q)$ of S^* at state q of $mp_{rp,k}(S^*)$ with respect to the canonical k-responsive events $\widehat{\Sigma}_\psi(q)$ at q.

We illustrate the calculation of the set $Cov_\psi(S^*)$ of all sets of responsive covering situations of service S^* with the following example.

Example 6.21. Consider a most permissive k-responsive controller $mp_{rp,k}(A_8^*)$ of the complete canonical k-responsive service $A_8^* = \mathcal{C}^2_{rp,k}(A_8)$ from Figure 6.7. For $k = 1$ and each state of the controller $mp_{rp,k}(A_8^*)$, the equivalence class of the canonical wait situations of A_8^* with respect to the canonical responsive events are illustrated as follows:

$\xi_{\widehat{\Sigma}}\widehat{\mathcal{K}}(q_1)$: $\{\{[7,[\,]]\}\}$ $\quad$ $\xi_{\widehat{\Sigma}}\widehat{\mathcal{K}}(q_2)$: $\{\{[10,[\,]]\}\}$

$\xi_{\widehat{\Sigma}}\widehat{\mathcal{K}}(q_3)$: $\{\{[18,[a]],[19,[a]]\}\}$ $\quad$ $\xi_{\widehat{\Sigma}}\widehat{\mathcal{K}}(q_4)$: $\{\{[18,[\,]],[19,[\,]]\}\}$

Then, the set $Cov_\psi(A_8^*)$ of all sets of responsive covering situations of A_8^* is given by $Cov_\psi(A_8^*) = \{\ \{[7,[\,]]\},\ \{[10,[\,]]\},\ \{[18,[a]],[19,[a]]\},\ \{[18,[\,]],[19,[\,]]\}\ \}$. ◁

In the next sections, we show how to employ sets of covering situations of the complete canonical responsive controller of the given service to decide its responsive equivalence.

6.3.3. Matching Responsive Covering Situations

In this section, we define a matching relation between two services by comparing specific subset of their situations. This comparison takes into account the sets of responsive covering situations of one service as described by Definition 6.20.

We first define the *responsive failures cover relation* of between two states as follows.

Definition 6.22 (Responsive failure cover of stable τ-strongly connected components).
Let S and T be two interface equivalent service automata. Let C_S be a stable τ-strongly connected component in S and C_T be a stable τ-strongly connected component in T. Then, component C_T of T *covers* a responsive failure of component C_S of S iff for each trace σ_S of S with $q_{0S} \stackrel{\sigma_S}{\Rightarrow}_S q_S$ and $q_S \in C_S$, there exists a trace σ_T of T with $q_{0T} \stackrel{\sigma_T}{\Rightarrow}_T q_T$ and $q_T \in C_T$ such that $\sigma_S = \sigma_T$ and $Refuse(Q_{C_S}) = Refuse(Q_{C_T})$ holds.

We illustrate the cover relation of stable τ-strongly connected components with the following example.

Example 6.23. Recall service A_8 from Figure 2.8 and a complete canonical k-responsive service $A_8^* = \mathcal{C}^2_{rp,k}(A_8)$ of A_8 from Figure 6.6. In Figure 6.8, the cover relations between states of A_8 and states of A_8^* are represented by dashed lines whereas states of A_8^* are represented by the shaded circle (the complete illustration of A_8^* is omitted). The stable τ-strongly connected components of A_8^* are represented by dotted rectangles, and each component contains exactly one stable state by construction.

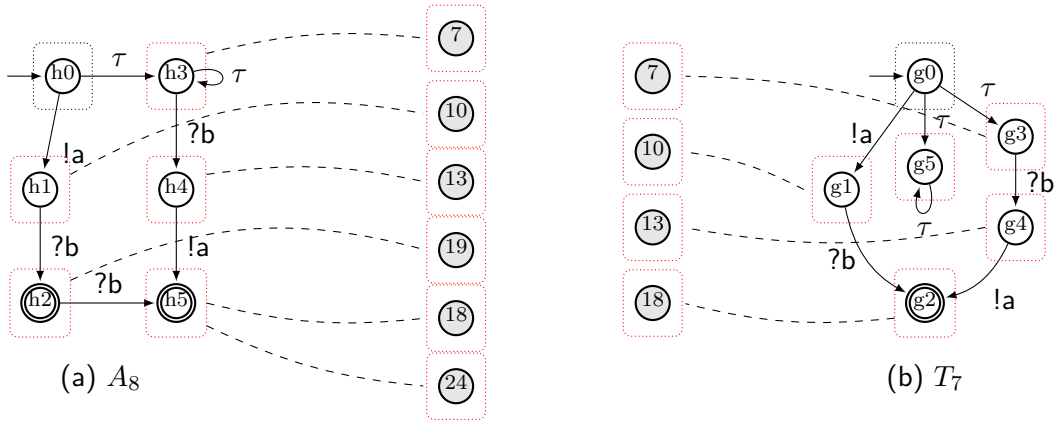


Figure 6.8.: Illustration of responsive failure covering relations of $Cov_{\psi}(\mathcal{C}_{rp,k}(A_8))$ from Example 6.21 by service A_8 from Figure 2.8 and service T_7 from Figure 6.10.

Consider service A_8. A τ-strongly connected component of A_8 is illustrated as a dotted rectangle. Every component of A_8 is a stable τ-strongly connected component except for the component containing $h0$. The component containing $h3$ has a responsive failure $[\epsilon, \{!a, \mathit{final}\}]$ and it covers the component of state 7 of A_8^*. The component containing $h1$ has a responsive failure $[?b, \{?b, \mathit{final}\}]$ and it covers the component of state 10 of A_8^*. The component containing $h2$ has a responsive failure $[!a?b, \{!a\}]$ and it covers the component of state 19 of A_8^*. The component containing $h4$ has a responsive failure $[?b, \{!a, \mathit{final}\}]$ and it covers the component of state 13 of A_8^*. The component containing $h5$ has two responsive failures $[!a?b?b, \{!a, ?b\}]$ and $[?b!, \{!a, ?b\}]$. Respectively, it covers the component of states 24 and the component of 19 of A_8^*. ◁

We use the responsive failures covering relation of τ-strongly connected component to define the covering relation between service T and the set $Cov_{\psi}(\mathcal{C}^2_{df,k}(S))$ of responsive covering situations of a complete canonical responsive service $\mathcal{C}^2_{rp,k}(S)$ of S. For this purpose, we define the following matching relation.

Definition 6.24 (Matching responsive covering situations).
Let S^* and T be two interface equivalent service automata. Let $\mu = \mathit{situations}(T)$ be a set of all situations of T with respect to the composition $T \oplus mp_{rp,k}(S^*)$ of T and a most permissive k-responsive controller $mp_{rp,k}(S^*)$ of S^* for message bound $k \in \mathbb{N}$ for each message channel.

Then, service T *covers* the set $Cov_{\psi}(S^*)$ of responsive covering situations of service S iff for each $e_S \in Cov_{\varphi}(S^*)$ there exists $[q_S, \mathcal{M}_S] \in e_S$ and $[q_T, \mathcal{M}_T] \in \mu$ such that

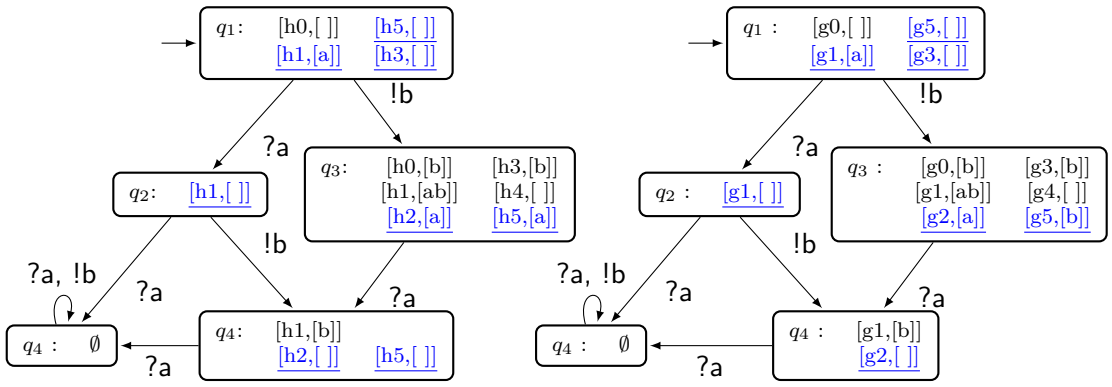
1. q_S is in a stable τ-strongly connected component C_S of S and q_T is in a stable τ-strongly connected component C_T of T,
2. C_T covers a responsive failure of C_S, and

3. $\mathcal{M}_S = \mathcal{M}_T$.

The set of all service automata that cover the set $Cov_\psi(S)$ of all sets of responsive covering situations of S is denoted by *rf-cover*(S).

In comparison to the matching between a service T and the set $Cov_\varphi(\mathcal{C}^2_{df,k}(S))$ of all sets of deadlock-free covering situations of service S, the matching relation defined in 6.24 checks for the cover relations between stable τ-strongly connected components instead of the cover relation between stable states. By construction, a complete canonical k-responsive controller of a given service does not have a τ-strongly connected component which contains more than one state.

Example 6.25. Consider message bound $k = 1$ on each message channel. Figure 6.9 illustrates the knowledge a most permissive k-responsive controller $mp_{df,k}(A^*_8)$ of $A^*_8 = \mathcal{C}^2_{df,k}(A_8)$ has (a) about service A_8 from Figure 2.8 and (b) about service T_7 from Figure 6.8. Each knowledge represents a partial view of the composition with service A_8 and service T_7 from the viewpoint of the controller $mp_{df,k}(A^*_8)$.



(a) with knowledge about service A_8 (b) with knowledge about service T_7

Figure 6.9.: The most permissive k-responsive controller $mp_{rp,k}(\mathcal{C}^2_{rp,k}(A_8))$ of $\mathcal{C}^2_{rp,k}(A_8)$: (a) with knowledge about service A_8 and (b) with knowledge about service T_7, for message bound $k = 1$ on each message channel.

Recall $Cov_\psi(A^*_8) = \{ \{[7, [\,]]\}, \{[10, [\,]]\}, \{[18, [a]], [19, [a]]\}, \{[18, [\,]], [19, [\,]]\} \}$ from Example 6.21 and the cover relations between states of A_8 and A^*_8 from Example 6.23.

Consider service A_8 and all its situations from Figure 6.9(a). For each $e \in Cov_\psi(A^*_8)$,

- $e = \{[7, [\,]]\}$, the component containing state $h3$ covers responsive failures of the component containing state 7 with a situation $[h3, [\,]]$ of A_8,
- $e = \{[10, [\,]]\}$, the component containing state $h1$ covers responsive failures of the component containing state 10 with a situation $[h1, [\,]]$ of A_8,

$e = \{[18,[a]],[19,[a]]\}$ and $e = \{[18,[\,]],[19,[\,]]\}$, the component containing state $h2$ covers responsive failures of the component containing state 19 with situations $[h2,[a]]$ and $[h5,[a]]$ of A_8, and the component containing state $h5$ covers responsive failures of the component containing state 18 with a situation $[h5,[\,]]$ of A_8.

Therefore, A_8 covers the set $Cov_\psi(A_8^*)$. ◁

The following theorem asserts that covering the set $Cov_\psi(\mathcal{C}^2_{rp,k}(S))$ of a complete canonical responsive service $\mathcal{C}^2_{rp,k}(S)$ of service S is a necessary condition to decide responsiveness equivalence of S.

Theorem 6.26 (Responsive failures cover of two responsively equivalent services).
For each message bound $k \in \mathbb{N}$ and each two interface equivalent services S and T:

$$T =_{rp,k} S \quad \Rightarrow \quad T \text{ covers } Cov_\psi(\mathcal{C}^2_{rp,k}(S))$$

where $\mathcal{C}^2_{df,k}(S)$ is a complete canonical k-deadlock-free service of S.

Proof. Let $S^* = \mathcal{C}^2_{rp,k}(S)$. Suppose $T =_{rp,k} S$. Because $S^* =_{rp,k} S$ (Lemma 5.35), $T =_{rp,k} S^*$ follows by transitivity.

Let $mp_{rp,k}(T)$ and $mp_{rp,k}(S^*)$ be most permissive k-responsive controllers of T and S^* respectively. Because $T =_{rp,k} S^*$, then $mp_{rp,k}(S^*)$ and $mp_{rp,k}(T)$ are bisimilar (Corollary 3.74). Consider state q_{mT} in $mp_{rp,k}(T)$ with $q_{0mT} \stackrel{\sigma_{\overline{m}}}{\Rightarrow} q_{mT}$ and state q_m in $mp_{rp,k}(S^*)$ with $q_{0m} \stackrel{\sigma_{\overline{m}}}{\Rightarrow} q_m$ where q_{0mT} is an initial state of $mp_{rp,k}(T)$ and q_{0m} is an initial state of $mp_{rp,k}(S^*)$ respectively.

Let $\widehat{\psi}(q_{mT})$ and $\widehat{\psi}(q_m)$ be the set of canonical responsive choices at q_{mT} and q_m respectively. Because $T =_{rp,k} S^*$, $\widehat{\psi}(q_m) = \widehat{\psi}(q_{mT})$ (Corollary 3.74).

Because $mp_{rp,k}(S^*)$ and $mp_{rp,k}(T)$ are deterministic and τ-free by construction, then their sets of canonical responsive event at q_m and at q_{mT} are equivalent, i. e., $\widehat{\Sigma}_\psi(q_m) = \widehat{\Sigma}_\psi(q_{mT})$. This means that the set $\widehat{\mathcal{K}}_\psi(q_m)$ of all canonical responsive situations at q_m and the set $\widehat{\mathcal{K}}_\psi(q_{mT})$ of all canonical responsive situations at q_{mT} preserve exactly the same set of canonical responsive choices.

Consider a partition $\xi_{\widehat{\Sigma}_\psi}(\widehat{\mathcal{K}}_\psi(q_m))$ of equivalence situations that describes all equivalent situations of S^* in which one situation can replace others without effecting the canonical responsive choices of $mp_{rp,k}(S^*)$.

Consider the two following cases :

– $\widehat{\mathcal{K}}_\psi(q_m) = \emptyset$: this implies either $\mathcal{K}(q_m) = \emptyset$ or $wait\mathcal{K}(q_m)) = \emptyset$. The first case $\mathcal{K}_\psi(q_m) = \emptyset$ means state q_m is not reachable in the composition $S^* \oplus mp_{rp,k}(S^*)$. In the latter case, assume $\mathcal{K}_\psi(q_m) \neq \emptyset$. This means every situation $[q,\mathcal{M}]$ in $\mathcal{K}(q_m)$ by definition is neither stable nor there is a stable τ-strongly connected component C of transition system of $\mathcal{K}_\psi(q_m)$ such that $[q,\mathcal{M}] \in Q_C$ For both cases, $\xi_{\widehat{\Sigma}_\psi(q_m)}(\widehat{\mathcal{K}}_\psi(q_m)) = \emptyset$ and there is no responsive covering situation of S^* associated with states q_m.

– $\widehat{\mathcal{K}}_\psi(q_m) \neq \emptyset$: Consider $e_S \in Cov_\psi(S^*)$ and the partition $\xi_{\widehat{\Sigma}_\psi(q_m)}(\widehat{\mathcal{K}}_\varphi(q_m))$ of canonical situations $\widehat{\mathcal{K}}_\psi(q_m)$ with respect to canonical events $\widehat{\Sigma}_\psi(q_m)$. This means that there exists a situation $[q_S, \mathcal{M}] \in e_S$ such that for each $[q'_S, \mathcal{M}'] \in e$ and for each $x \in \widehat{\Sigma}_\psi(q_m)$ holds : $activ_k([q_S, \mathcal{M}], x) \Leftrightarrow activ_k([q'_S, \mathcal{M}'], x)$ by definition.

As the situation $[q_S, \mathcal{M}]$ of S^* is a wait situation in $\mathcal{K}_\psi(q_m)$, then there exists a stable τ-strongly connected component C in S^* such that $[\sigma, X] \in$ *rp-failures*(S^*) with $q_{0S} \stackrel{\sigma}{\Rightarrow} q_S$, $q_S \in Q_C$, and $X =$ *Refuse*(Q_C). As $T =_{rp,k} S$ implies $T \sqsubseteq_{rp,k} S$, *rp-failures*$(S^*) \supseteq$ *rp-failures*(T) follows (Theorem 5.36).

Suppose T is derived from S^* by removing each path $q_{0S} \stackrel{\sigma'}{\Rightarrow} q'_S$ that leads to a stable τ-strongly connected component C containing q'_S for every $[q'_S, \mathcal{M}'] \in e_S$ including the subsequent path that follows C. This means T does not cover $Cov_\psi(S^*)$ as there exists $q_S \in e_S \in Cov_\psi(S^*)$ such that none of the stable τ-strongly connected component of T can cover the stable τ-strongly connected component C.

As $T =_{rp,k} S =_{rp,k} S^*$, we have $q_{0mT} \stackrel{\sigma_m}{\Rightarrow} q_{mT}$, $q_{0m} \stackrel{\sigma_m}{\Rightarrow} q_m$, and $\widehat{\psi}(q_m) = \widehat{\psi}(q_{mT})$. It follows that there exists $x \in \widehat{\Sigma}_\psi(q_m)$ with $activ_k([q_S, \mathcal{M}], x) \Leftrightarrow activ_k([q'_S, \mathcal{M}'], x)$ but $x \notin \widehat{\Sigma}_\psi(q_{mT})$ as all paths from S^* that result in a situation $[q_S, \mathcal{M}] \in e_S$ have been removed in order to derive T. This contradicts to the assumption $\widehat{\Sigma}_\psi(q_m) = \widehat{\Sigma}_\psi(q_{mT})$.

Therefore, for each $e_S \in Cov_\psi(S^*)$ there must be a state $q_S \in e_S$ that is contained in a stable τ-strongly connected component C, and a stable τ-strongly connected component C_T that contains state q_T with $q_{0T} \stackrel{\sigma}{\Rightarrow}_T q_T$ and *Refuse*$(Q_C) =$ *Refuse*(q_{C_T}).

In case of cyclic behavior where there exist σ'_S that revisits q_S with $q_S \stackrel{\sigma'_S}{\Rightarrow}_S q_S$, there must be a stable τ-strongly connected component C'_T that contains state q'_T with $q_T \stackrel{\sigma'_T}{\Rightarrow}_T q'_T$ and state $q'_S \in e_S$ is contained in a stable τ-strongly connected component C'_S with $q_S \stackrel{\sigma'_S}{\Rightarrow}_T q'_S$ such that $\sigma'_S = \sigma'_T$ and *Refuse*$(Q_{C'_S}) =$ *Refuse*$(Q_{C'_T})$.

Thus, we conclude that T covers $Cov_\psi(S^*)$. □

Theorem 6.26 suggests a necessary condition for deciding if two services are responsively equivalent. Nevertheless, the reverse of Theorem 6.26 does not necessarily hold.

Similarly to the characterization of deadlock freedom equivalence, covering the set of all sets of responsive covering situations is not sufficient to decide responsive equivalence of a given service. As service T covers the set $Cov_\psi(\mathcal{C}^2_{rp,k}(S))$ of $\mathcal{C}^2_{rp,k}(S)$ of service S only means that T covers the components that preserve all canonical choices of all responsive controllers of S, but it does not guarantee that behavior of T will responds to every responsive controller of service S.

Example 6.27. Consider service T_7 from Figure 6.8 and a complete canonical k-responsive substitute $A^*_8 = \mathcal{C}^2_{rp,k}(A_8)$ of A_8 from Figure 6.6 for message bound $k = 1$ on each message channel. Service T_7 has interface $I = \{b\}$ and $O = \{a\}$, same interface as service A_8 from Figure 2.8. In Figure 6.8, the cover relations between states of T_7 and states of A^*_8 are represented by dashed lines whereas states of A^*_8 are represented by the shaded circle (the complete illustration of A^*_8 structure of is omitted).

Consider service T_7. Every component of T_7 is a stable τ-strongly connected component except the one containing $g0$. Each stable τ-strongly connect component can cover the stable τ-strongly connected component of A_8^* except the component containing g_5.

Recall the set $Cov_\psi(A_8^*) = \{\ \{[7,[\ ]]\}, \{[10,[\ ]]\}, \{[18,[a]],[19,[a]]\}, \{[18,[\ ]],[19,[\ ]]\}\ \}$ from Example 6.21.

Consider all its situations of T_7 with respect to the composition $T_7 \oplus mp_{rp,k}(A_8^*)$ from Figure 6.9(b). For each $e \in Cov_\psi(A_8^*)$, we have can find a component of T_7 containing state q_{T_7} that covers responsive failure of the component containing state q for some $[q, \mathcal{M}] \in e$ with a situation $[q_{T_7}, \mathcal{M}]$ of $mp_{rp,k}(A_8^*)$. Therefore, T_7 covers the set $Cov_\psi(A_8^*)$. Nevertheless, T_7 is not k-responsively controllable, as it is possible for T_7 to perform an unbroken infinite sequence of internal τ events due to the component containing state $g5$, and therefore, it may not respond to any interacting service afterwards. That is, T_7 is not a substitute for A_8 under k-responsive equivalence. $\lhd$

In the next section, we show sufficient conditions for deciding responsiveness equivalence of a given service S. That is, service S and T are responsively equivalent whenever T refines $\mathcal{C}^2_{rp,k}(S)$ under responsive failures and T covers the set $Cov_\psi(S^*)$ of all sets of responsive covering situations of $\mathcal{C}^2_{rp,k}(S)$.

6.3.4. Responsive Equivalence Guidelines

In this section, we prove necessary and sufficient conditions for deciding responsiveness equivalence of a given service. In the following theorem, we show that refining a canonical responsive service under responsive failures and covering the set of all sets of responsive covering situations of a canonical responsive service together is necessary and sufficient to decide responsive equivalence of a given service.

Theorem 6.28 (Characterizing all responsively equivalent services).
For each message bound $k \in \mathbb{N}$ and each two k-responsively controllable service S and T:

$$rp_k Equiv(S) \quad = \quad rf\text{-}refine(\mathcal{C}^2_{rp,k}(S) \ \cap \ rf\text{-}cover(\mathcal{C}^2_{rp,k}(S))$$

where $\mathcal{C}^2_{rp,k}(S)$ is a complete canonical k-responsive service of S.

Proof. Let $S^* = \mathcal{C}^2_{rp,k}(S)$. We prove this theorem in two directions.

$\Rightarrow$: Suppose $T =_{rp,k} S$. This means, $T \in rf\text{-}refine(S^*)$ (Theorem 5.36) and $T \in rf\text{-}cover(S^*)$ (Theorem 6.26).

$\Leftarrow$: Suppose $S^* \sqsubseteq_{RF} T$ and T covers $Cov_\psi(S^*)$.

Because $S^* \sqsubseteq_{RF} T$, $T \sqsubseteq_{rp,k} S =_{rp,k} S^*$ follows (Theorem 5.36 and Lemma 5.35). Let $mp_{rp,k}(T)$ and $mp_{rp,k}(S^*)$ be most-permissive responsive controllers of T and S^* respectively. We will show that (1) $mp_{rp,k}(S^*)$ and $mp_{rp,k}(T)$ are bisimilar and (2) for each pair $[q_{mT}, q_{mS^*}]$ of bisimilar states holds: $\psi(q_{mT}) = \psi(q_{mS^*})$.

1. Because $T \sqsubseteq_{rp,k} S^*$, there exists a strong simulation relation ϱ such that $mp_{rp,k}(T)$ simulates $mp_{rp,k}(S^*)$ with ϱ (Proposition 3.67).

   By construction, $mp_{rp,k}(S^*)$ and $mp_{rp,k}(T)$ are derived from the initial situations of S^* and T using the closure and event operations on knowledge of $mp_{rp,k}(S^*)$ and $mp_{rp,k}(T)$ respectively. This means that the initial state of $mp_{rp,k}(S^*)$ is $[q_{0S^*}, [\,]] \in \mathcal{K}(q_{0mS^*})$ and the initial state of $mp_{rp,k}(T)$ is $[q_{0T}, [\,]] \in \mathcal{K}(q_{0mT})$.

   Let $[q_{0mT}, q_{0mS^*}] \in \varrho^*$ be a binary relation between states of $mp_{rp,k}(T)$ and $mp_{rp,k}(S^*)$.

   Consider $[q_{mT}, q_{mS^*}] \in \varrho^*$ with $[q_{S^*}, \mathcal{M}_{S^*}] \in \mathcal{K}(q_{mS^*})$ and $q_{S^*} \in e \in Cov_\psi(S^*)$ with $q_{0S^*} \overset{\sigma_*}{\Rightarrow} q_{S^*}$, a stable τ-strongly connected component C that contains q_{S^*} and $Refuse(Q_C) = X^*$.

   Because *rp-failures*$(S^*) \supseteq$ *rp-failures*(T) holds by assumption, then for each $[\sigma_T, X_T] \in$ *rp-failures*(T) there is a $[\sigma_{S^*}, X_{S^*}] \in$ *rp-failures*(S^*) with $\sigma_{S^*} = \sigma_T$ and $X_{S^*} = X_T$.

   Because T covers the set $Cov_\psi(S^*)$, by definition there exists a stable τ-strongly connected component C_T that covers a stable τ-strongly connected component C containing state q_{S^*}. This implies, (1) for each $[q_{S^*}, \mathcal{M}_{S^*}] \in e$, T has the same failure as S^* and (2) in case there exists a sequence of events starting from a state in C and revisiting C for each round, there exists also the same sequence of events starting from a state in C_T to a stable τ-strongly connected component C'_T such that component C and component C'_T refuse exactly the same set of events.

   For each $m \in \Sigma$, the successor state of state q_{mT} is uniquely determined by $\mathcal{K}'(q_{mT}) = closure(event(\mathcal{K}(q_{mT})), m)$ of $mp_{rp,k}(T)$ by construction. Because S^* and T have the same set of input I and output O message channels as well as the set of all situations of S^* and that of T are both bounded by k, then it follows that there exists also the successor state of state q_{mT} that is uniquely determined by $\mathcal{K}'(q_{mS^*}) = closure(event(\mathcal{K}(q_{mS^*})), m)$ of $mp_{rp,k}(S^*)$ by construction. This is because the failures that are not offered by T do not affect the equivalence classes of wait situations at $\mathcal{K}(q_{mT})$.

   As $mp_{rp,k}(S^*)$ and $mp_{rp,k}(T)$ are interface equivalent and deterministic, the relation ϱ^* is a strong simulation relation such that $mp_{rp,k}(S^*)$ strongly simulates $mp_{rp,k}(T)$ with ϱ^*.

   As $mp_{rp,k}(T)$ also strongly simulates $mp_{rp,k}(S^*)$ with ϱ and $mp_{rp,k}(S^*)$ and $mp_{rp,k}(T)$ are deterministic, this means that $mp_{rp,k}(S^*)$ and $mp_{rp,k}(T)$ are bisimilar.

2. Consider a pair $[q_{mT}, q_{mS^*}]$ of bisimilar states of $mp_{rp,k}(S^*)$ and $mp_{rp,k}(T)$. It follows from bisimulation that $enable(q_{mT}) = enable(q_{mS^*})$.

   Consider $\mathcal{K}(q_{mS^*})$ and $\mathcal{K}(q_{mT})$. Because $mp_{rp,k}(S^*)$ is a most permissive k-responsive controller of S^* and $mp_{rp,k}(T)$ is a most permissive k-responsive

controller of T, this means that $\mathcal{L}_k(\mathbb{E}_f, \mathcal{K}(q_{mS^*})) = \mathcal{L}_k(\mathbb{E}_f, \mathcal{K}(q_{mT}))$ follows from $enable(q_{mT}) = enable(q_{mS^*})$ where $\mathbb{E}_f = \Sigma \cup \{final\}$.

Consider $\mathcal{K}(q_{mS^*})$ and the following cases:

- $\mathcal{K}(q_{mS^*}) = \emptyset$: this means that state q_{mS^*} is not reachable in the composition of S^* with any of its controllers. By construction, $\psi(q_{mS^*}) \subseteq \Sigma \cup \{final\}$. As (q_{mT}, q_{mS^*}) is a pair of bisimilar states, this means $\mathcal{K}(q_{mT}) = \emptyset$ follows and q_{mT} is also not reachable in the composition of T with any of its controllers. As every responsive failure of T is also a responsive failure of S^* by assumption, there is no responsive failure of T that is not described by S^*. That is, $\psi(q_{mT}) = \psi(q_{mS^*})$ holds.
- $\mathcal{K}(q_{mS^*}) \neq \emptyset$ and $wait(\mathcal{K}(q_{mS^*})) = \emptyset$: this means, every corresponding state of q_{mS^*} in S^* can make an internal move without help from a controller of S^* and never perform an external event. By construction, $\psi(q_{mS^*})$ holds and state q_{mS^*} is being removed during the construction of $mp_{rp,k}(S^*)$. Similarly for state q_{mT} in $mp_{rp,k}(T)$.
- $\mathcal{K}(q_{mS^*}) \neq \emptyset$ and $wait(\mathcal{K}(q_{mS^*})) \neq \emptyset$:

  Because $S^* \sqsubseteq_{RF} T$, then $T \sqsubseteq_{rp,k} S^*$ holds (Theorem 5.36). Therefore, $[q_{mT}, q_{mS^*}] \in \varrho$ is a strong simulation relation and $\psi(q_{mS^*}) \subseteq \psi(q_{mT})$ holds (Lemma 3.70).

  Let $ch \in \psi(q_{mT})$, we will show that $ch \in \psi(q_{mS^*})$ holds.

  Let $q_{S^*} \in e_S \in Cov_\psi(S^*)$. Consider $q_{0S^*} \stackrel{\sigma}{\Rightarrow} q_{S^*}$ with a stable τ-strongly connected component C that contains q_{S^*} with $X = Refuse(Q_C)$.

  Because T covers $Cov_\psi(S^*)$, there exists a state q_T in T that covers state q_{S^*} by definition. This implies, (1) for each $[q_{S^*}, \mathcal{M}_{S^*}]$, T has the same failure as S^* and (2) in case there exists a sequence of events starting from a state in C and revisiting C for each round, there exists also the same sequence of events starting from a state in C_T to a stable τ-strongly connected component C'_T such component C and component C'_T refuse exactly the same set of events.

  Consider a pair $[q_{mT}, q_{mS^*}]$ of bisimilar states with $[q_T, \mathcal{M}_T] \in wait(\mathcal{K}(q_{mT}))$ and $[q_{S^*}, \mathcal{M}^*] \in wait(\mathcal{K}(q_{mS^*}))$.

  Because $ch \in \psi(q_{mT})$, then there exists $m \in ch$ such that $activ_k([q_T, \mathcal{M}], m) = true$ holds. Because $\mathcal{L}_k(\mathbb{E}_f, \mathcal{K}(q_{mS^*})) = \mathcal{L}_k(\mathbb{E}_f, \mathcal{K}(q_{mT}))$ and $Refuse(q_C) = Refuse(q_{C_T}) = X$, $activ_k([q_{S^*}, \mathcal{M}^*], m) = true$ also holds. This means, $ch \in \psi(q_{mS^*})$ follows.

  Therefore, $\psi(q_{mS^*}) \supseteq \psi(q_{mT})$ holds.

As $mp_{rp,k}(S^*)$ and $mp_{rp,k}(T)$ are bisimilar and for each pair $[q_{mT}, q_{mS^*}]$ of bisimilar states holds: $\psi(q_{mT}) = \psi(q_{mS^*})$, thus, $T =_{rp,k} S$ holds (Corollary 3.71).

Thus, the theorem holds. □

The value of Theorem 6.28 is for characterizing all services that are responsively equivalent to a given service S. That is, two services S and T are responsively equivalent whenever (1) T *refines* the canonical responsive service $\mathcal{C}^2_{rp,k}(S)$ of S under responsive failures and (2) T *covers* the set $Cov_\psi(\mathcal{C}^2_{rp,k}(S))$ of all sets of responsive covering situations of the canonical responsive service $\mathcal{C}^2_{rp,k}(S)$ of S.

We present the *responsive equivalence guidelines* as a finite representation of all k-responsively equivalent services of a given service S.

Definition 6.29 (Responsive equivalence guideline).
Let $k \in \mathbb{N}$ be a message bound for each message channel and $\mathcal{C}^2_{rp,k}(S)$ be the complete canonical k-responsive service of a k-responsively controllable service automaton S. Let $Cov_\psi(\mathcal{C}^2_{rp,k}(S))$ be the set of all sets of responsive covering situations of $\mathcal{C}^2_{rp,k}(S)$. Then, the *responsive equivalence guidelines* of service S is denoted by a pair $[\mathcal{C}^2_{rp,k}(S), Cov_\psi(\mathcal{C}^2_{rp,k}(S))]$ of $\mathcal{C}^2_{rp,k}(S)$ and $Cov_\psi(\mathcal{C}^2_{rp,k}(S))$.

A service T *matches* with a responsive equivalence guidelines $[\mathcal{C}^2_{rp,k}(S), Cov_\psi(\mathcal{C}^2_{rp,k}(S))]$ of service S iff T refines $\mathcal{C}^2_{rp,k}(S)$ under stable failures and T covers $Cov_\psi(\mathcal{C}^2_{rp,k}(S))$.

For characterizing the set of all responsively equivalent substitutes for a given service S, it is not possible to replace the complete canonical responsive substitution service $\mathcal{C}^2_{rp,k}(S)$ of S in the equivalence guidelines of S by a compact canonical responsive substitute $\widehat{\mathcal{C}}^2_{rp,k}(S)$ for S. This is because, by construction, the compact canonical responsive substitute $\widehat{\mathcal{C}}^2_{rp,k}(S)$ for S does not encode within its structure all possible responsive choices of controllers of S, but only essential choices called canonical responsive choices. Similar to the construction of deadlock-free equivalence guidelines of S, we can optimize the construction procedure of the service $\mathcal{C}^2_{rp,k}(S)$ by employing the compact canonical responsive controller $\widehat{\mathcal{C}}_{rp,k}(S)$ of S instead of the complete canonical responsive controller $\mathcal{C}_{rp,k}(S)$ of S. We discuss more on this issue in Chapter 7.

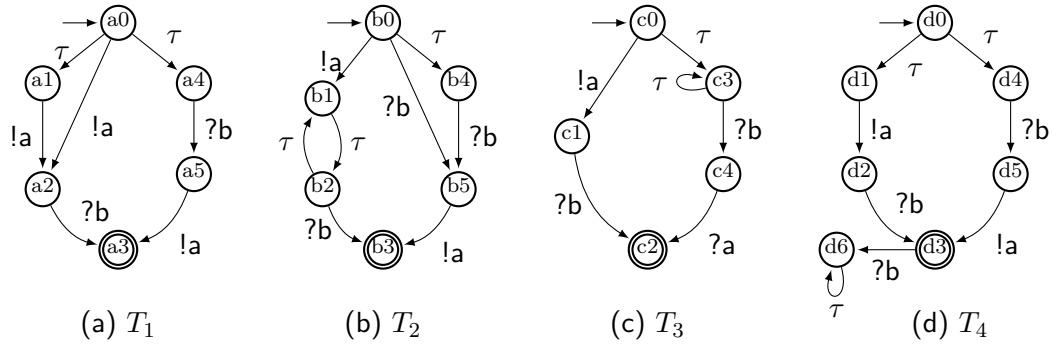
We illustrate how responsive equivalence guidelines characterize the set of responsively equivalent services with the following examples.

Example 6.30. Consider the service A_8 from Figure 2.8 and seven services T_1, T_2, T_3, T_4, T_5, T_6, T_7, T_8, and T_9 from Figure 6.10. All have the same interface $I = \{b\}$ and $O = \{a\}$. For message bound $k = 1$ on each message channel, a k-responsive equivalence guidelines of service A_8 is illustrated by $[\mathcal{C}^2_{rp,k}(A_8), Cov_\psi(\mathcal{C}^2_{rp,k}(A_8))]$ from Example 6.21.

The four services T_1, T_2, T_3, and T_4 are equivalent to service A_8 under responsiveness and they are characterized by the equivalence guidelines of A_8. The five services T_5, T_6, T_7, T_8, and T_9 are not equivalent to service A_8 under responsiveness, and they are not characterized by the equivalence guidelines of A_8.

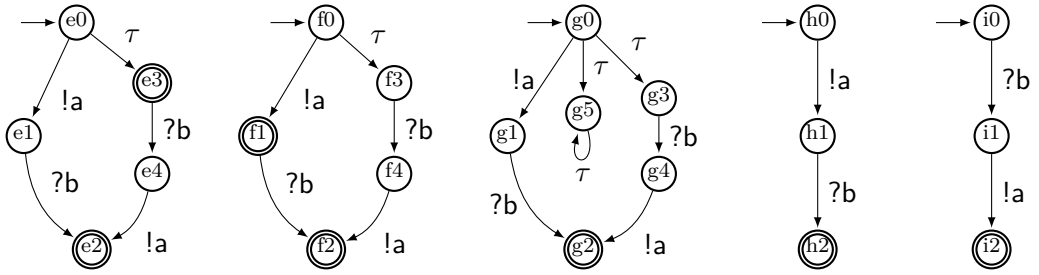
We see that each service from Figure 6.10 except for service T_7 refines the service $\mathcal{C}^2_{rp,k}(A_8)$ under responsive failures. There is one stable τ-strongly connected component of T_7 that contains state $g5$ which refuses every event in $\{!a, ?b, \mathit{final}\}$. This means, there is a responsive failure $[\epsilon, \{!a, ?b, \mathit{final}\}]$ of T_7 that is not described by a responsive failure

Positive Instances characterized by an equivalence guideline of A_8 *for* $k = 1$



(a) T_1 (b) T_2 (c) T_3 (d) T_4

Negative Instances characterized by an equivalence guideline of A_8 *for* $k = 1$



(e) T_5 (f) T_6 (g) T_7 (h) T_8 (i) T_9

Figure 6.10.: Nine services: T_1, T_2, T_3, T_4, T_5, T_6, T_7, T_8, and T_9; each with $I = \{b\}$ and $O = \{a\}$ in comparison the responsive equivalence guidelines $[\mathcal{C}^2_{rp,k}(A_8), Cov_\psi(\mathcal{C}^2_{rp,k}(A_8))]$ of A_8 from Figure 2.8 for message bound $k = 1$ on each message channel.

of the service $\mathcal{C}^2_{rp,k}(A_8)$. Therefore, every service except service T_7 is a substitute for service A_8 under k-responsiveness as asserted by Theorem 5.32.

Services T_5 and T_8 are not equivalent to service A_8 under k-responsiveness. This is because there is the set $e_1 \in Cov_\psi(\mathcal{C}^2_{rp,k}(A_8))$ with $e_1 = \{[7, [\,]]\}$ and neither T_5 nor T_8 can cover the responsive failure $[\epsilon, \{!a, \mathit{final}\}]$ described by state 7 of $\mathcal{C}^2_{rp,k}(A_8)$.

Services T_6 and T_9 are not equivalent to service A_8 under k-responsiveness. This is because there is the set $e_2 \in Cov_\psi(\mathcal{C}^2_{rp,k}(A_8))$ with $e_2 = \{[10, [\,]]\}$ and neither T_6 nor T_9 can cover the responsive failure $[!a, \{!a, \mathit{final}\}]$ described by state 10 of $\mathcal{C}^2_{rp,k}(A_8)$. $\lhd$

In the following example, we illustrate the calculation of an equivalence guideline for a service that contains a cycle.

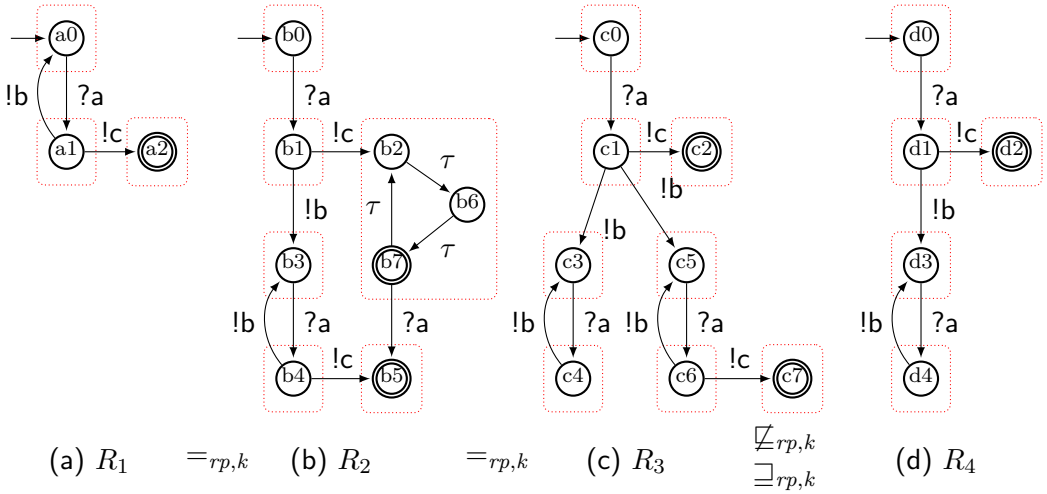


Figure 6.11.: Four services: R_1, R_2, R_3, and R_4; each with interface $I = \{a\}$ and $O = \{b, c\}$. The three services R_1, R_2, and R_3 are equivalent, whereas R_4 is not equivalent to the three services under k-responsiveness for message bound $k = 1$ on each message channel.

Example 6.31. Figure 6.11 illustrates four services R_1, R_2, R_3, and R_4. All have the same interface $I = \{b, c\}$ and $O = \{a\}$. For message bound $k = 1$ on each message channel, we assume a canonical k-responsive controller $\mathcal{C}_{rp,k}(R_1)$ of R_1 (omitted) and its k-responsive operating guideline $mp_{rp,k}(\mathcal{C}_{rp,k}(R_1))^{\psi^*}$ illustrated in Figure 6.12. Figure 6.12 also illustrates a complete canonical k-responsive service $R_1^* = \mathcal{C}^2_{rp,k}(R_1)$ of R_1 constructed from the operating guidelines $mp_{rp,k}(\mathcal{C}_{rp,k}(R_1))^{\psi^*}$.

Every service in Figure 6.11 refines R_1^* under responsive failures. This means, each one is a substitute for R_1 under k-responsiveness inclusion. However, only R_1, R_2, and R_3 are equivalent under k-responsiveness, but R_4 is not equivalent to any other.

Consider a most permissive k-responsive controller $mp_{rp,k}({R_1}^*)$ of R_1^* from Figure 6.13. The underlined situations denote the wait situations of R_1^*.

At state q_1 of $mp_{rp,k}(R_1^*)$, the set of all canonical valid responsive choices is $\widehat{\psi^*}(q_1)$ $= \{\{!a\}\}$ and the set of all canonical events is $\widehat{\Sigma}_{\psi^*}(q_2) = \{!a\}$. To calculate $\widehat{\mathcal{K}}(q_1)$, we consider the activation of events for wait situations at q_1 in the following table.

$[q, \mathcal{M}] \in wait(\mathcal{K}(q_1))$	[2, [ ]]	[3, [ ]]
$activ_k([q, \mathcal{M}], !a)$	*true*	*true*
$activ_k([q, \mathcal{M}], ?b)$	*false*	*false*
$activ_k([q, \mathcal{M}], ?c)$	*false*	*false*
$activ_k([q, \mathcal{M}], final)$	*false*	*true*
$enable(q)$	$\{?a\}$	$\{?a, final\}$

Situation $[3, [\,]]$ is not a canonical situation in $\widehat{\psi^*}(q_1)$. This is because $activ_k([3, [\,]], final)$ $= true$, but *final* is not a canonical event at state q_1 (*i. e.*, $final \notin \widehat{\Sigma}_{\psi^*}(q_1)$). Therefore,

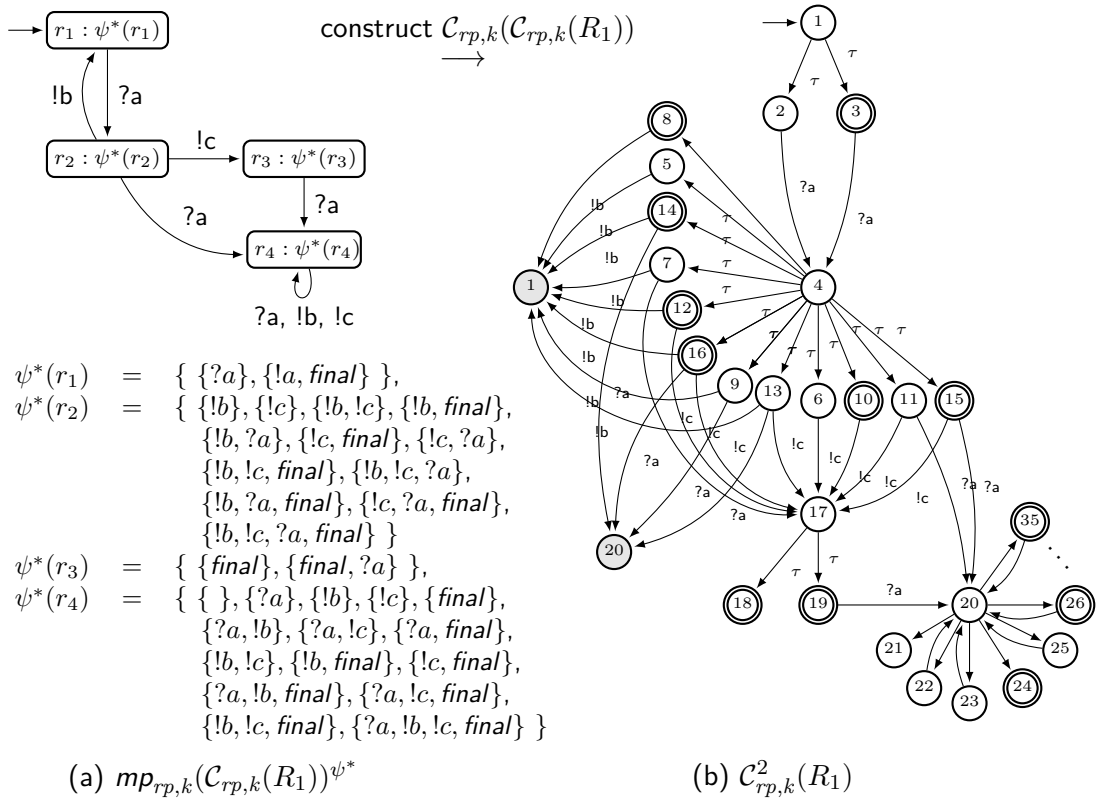


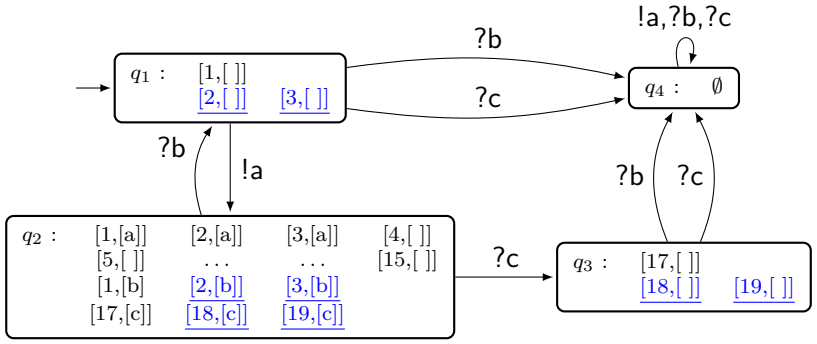
Figure 6.12.: A k-responsive operating guideline $mp_{rp,k}(\mathcal{C}_{rp,k}(R_1))^{\psi^*}$ of $\mathcal{C}_{rp,k}(R_1)$ (omitted) and a complete canonical k-responsive service; $\mathcal{C}^2_{rp,k}(R_1)$ of R_1 from Figure 6.11, both with $I = \{a\}$ and $O = \{b, c\}$ for message bound $k = 1$.

$[2, [\,]]$ is the only canonical wait situation at state q_1.

At state q_2 of $mp_{rp,k}(R_1^*)$, the set of all canonical choices is $\widehat{\psi^*}(q_2) = \{\{?b\}, \{?c\}\}$ and the set of all events described by $\widehat{\psi^*}(q_2)$ is $\widehat{\Sigma}_{\psi^*}(q_2) = \{?b, ?c\}$. To calculate $\widehat{\mathcal{K}}(q_2)$, we consider the activation of events for wait situations at q_2 in the following table.

$[q, \mathcal{M}] \in wait(\mathcal{K}(q_2))$	[2, [b]]	[3, [b]]	[18, [c]]	[19, [c]]
$activ_k([q, \mathcal{M}], !a)$	*true*	*true*	*true*	*true*
$activ_k([q, \mathcal{M}], ?b)$	*true*	*true*	*false*	*false*
$activ_k([q, \mathcal{M}], ?c)$	*false*	*false*	*true*	*true*
$activ_k([q, \mathcal{M}], final)$	*false*	*false*	*false*	*false*
$enable(q)$	$\{?a\}$	$\{?a, final\}$	$\{final\}$	$\{final, ?a\}$

Every wait situation in q_2 is a canonical situation. Though each wait situation $[q, \mathcal{M}]$ activates event $!a$, i. e., $activ_k([q, \mathcal{M}], final) = true$. However, $!a$ is not a responsive choice at q_2. Event $!a$ is also not a legally updated event at state q_2, as $!a$ at state q_2 definitely violates message bound $k = 1$ on each message channel due to situation $[1, [a]]$ for example. Therefore, $\widehat{\mathcal{K}}(q_1) = \{\{[2, [b]], [3, [b]]\}, \{[18, [c], [19, [c]]\}\}$.



$$
\begin{array}{lcl}
\psi^*(r_1) & = & \{\ \{!a\},\{!a,?b\},\{!a,?c\},\{!a,\mathit{final}\},\{!a,?b,?c\},\{!a,?b,\mathit{final}\},\\
 & & \quad \{!a,?b,?c,\mathit{final}\}\ \}\\
\psi^*(r_2) & = & \{\ \{?b,?c\},\{?b,?c,\mathit{final}\}\ \}\\
\psi^*(r_3) & = & \{\ \{\mathit{final}\},\{\mathit{final},?b\},\{\mathit{final},?c\},\{\mathit{final},?b,?c\}\ \}\\
\psi^*(r_4) & = & \{\ \{\ \},\{!a\},\{?b\},\{?c\},\{\mathit{final}\},\{!a,?b\},\{!a,?c\},\{!a,\mathit{final}\},\\
 & & \quad \{?b,?c\},\{?b,\mathit{final}\},\{?c,\mathit{final}\},\{!a,?b,\mathit{final}\},\{!a,?c,\mathit{final}\},\\
 & & \quad \{?b,?c,\mathit{final}\},\{!a,?b,?c,\mathit{final}\}\ \}
\end{array}
$$

Figure 6.13.: A k-responsive operating guideline $mp_{rp,k}(\mathcal{C}^2_{rp,k}(R_1))^{\psi^*}$ of a canonical k-responsive substitute $\mathcal{C}^2_{rp,k}(R_1)$ (from Figure 4.3) and its responsive choices $\psi^*(q)$ at each state q.

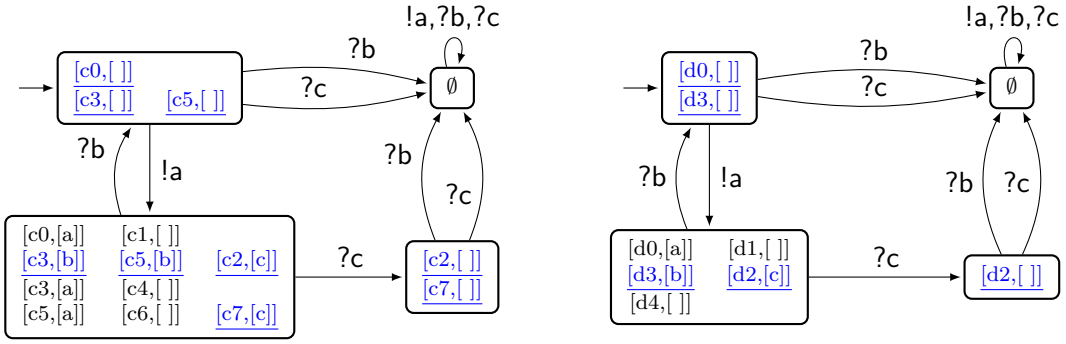
At state q_3 of $mp_{rp,k}(R_3^*)$, the set of all canonical choices is $\widehat{\varphi}(q_3) = \{\{\mathit{final}\}\}$ and the set of all events described by $\widehat{\varphi}(q_3)$ is $\widehat{\Sigma}_\varphi(q_3) = \{\mathit{final}\}$. To calculate $\widehat{\mathcal{K}}(q_3)$, we consider the activation of events for wait situations at q_3 in the following table.

$[q,\mathcal{M}] \in \mathit{wait}(\mathcal{K}(q_3))$	[18, [ ]]	[19, [ ]]
$\mathit{activ}_k([q,\mathcal{M}],!a)$	*true*	*true*
$\mathit{activ}_k([q,\mathcal{M}],?b)$	*false*	*false*
$\mathit{activ}_k([q,\mathcal{M}],?c)$	*false*	*false*
$\mathit{activ}_k([q,\mathcal{M}],\mathit{final})$	*true*	*true*
$\mathit{enable}(q)$	$\{\mathit{final}\}$	$\{?a,\mathit{final}\}$

Both situations are canonical situations in $\widehat{\psi^*}(q_3)$. Though each wait situation $[q,\mathcal{M}]$ activates event $!a$, i. e., $\mathit{activ}_k([q,\mathcal{M}],\mathit{final}) = \mathit{true}$, and $!a$ is a legally updated event at state q_3. However, the choice $\{!a\}$ is not a $\widehat{\varphi}(q_3)$ responsive choice of q_3. Therefore, $!a$ is not a canonical responsive event at q_3 and $\widehat{\mathcal{K}}(q_3) = \{[18,[\ ],[19,[\ ]]\}$.

Then, the set $Cov_\psi(R_1^*)$ of all sets of responsive covering situations of R_1^* is illustrated by $Cov_\psi(R_1^*) = \{\ \{[2,[\ ]]\},\{\{2,[b]],[3,[b]]\},\{[18,[c],[19,[c]]\}\},\{[18,[\ ],[19,[\ ]]\}\ \}$. The k-responsive equivalence guideline of service R_1 is illustrated by $[R_1^*, Cov_\psi(R_1^*)]$.

Consider service R_4 from Figure 6.11 and all its situations from Figure 6.14. A τ-strongly connected component of R_4 is illustrated as a dotted rectangle. Every component of R_4 is a stable τ-strongly connected component. The component containing state $d0$



(a) with knowledge about service R_3

(b) with knowledge about service R_4

Figure 6.14.: The most permissive k-responsive controller $mp_{rp,k}(\mathcal{C}^2_{rp,k}(R_1))$ of $\mathcal{C}^2_{rp,k}(R_1)$: (a) with knowledge about service R_3 and (b) with knowledge about service R_4, for message bound $k = 1$ on each message channel.

and the component containing state $d3$ cover the component of state 2 of R_1^*. However, neither $[d2, [c]]$ nor $[d3, [b]]$ is a situation of R_4. This means, none of situations of R_4 can cover responsive failures described by $\{[2, [b]], [3, [b]]\} \in Cov_\psi(R_1^*)$. Therefore, R_4 cannot cover $Cov_\psi(R_1^*)$. Though not equivalent to R_1 under k-responsiveness, R_4 is a substitute for R_1 under k-responsiveness inclusion. The circumstance in which R_4 misses such situations allows R_4 to interact with at least one additional responsive controller, that is, a service that does not receive message c after it send out the second message a. Obviously, this service is not a controller of R_1.

Consider service R_3 from Figure 6.11 and all its situations from Figure 6.14. A τ-strongly connected component of R_3 is illustrated as a dotted rectangle. Every component of R_3 is a stable τ-strongly connected component. The component containing state $c0$ has a responsive failure $[\epsilon, \{!b, !c, final\}]$ and it covers the compoent of state 2 of R_1^*. In contrast to R_4, there is a stable τ-strongly connected component C_{c2} of R_3 containing state $c2$ which refuses the same set $\{?a, !b, !c\}$ as the component of state 18 of R_1^*. The component C_{d2} can cover the failure of the component of state 18. Though R_3 can reach state $c3$ after performing $!b$ and from this point on it is no longer possible for R_3 to perform $!c$. Nevertheless, this behavior of R_3 does not affect a controller of R_1, as a controller of R_1 must be able to receive both messages b and c after sending out message a. Therefore, service R_3 covers the set $Cov_\psi(R_1^*)$ of R_1^*. ◁

6.4. Concluding Remarks

In this chapter, we characterized the set of all equivalent substitutes for a given service S under two behavioral compatibility criteria $\mathcal{B} \in \{df_k, rp_k\}$. For each behavioral compatibility criterion $\mathcal{B}$, we proposed a $\mathcal{B}$ equivalence guideline of service S as a finite representation of all equivalent substitutes for service S under $\mathcal{B}$. Several underlying

concepts and techniques for constructing a $\mathcal{B}$ equivalence guideline are similar for each behavioral compatibility criterion $\mathcal{B} \in \{df_k, rp_k\}$, though their subtle differences of details require separate attentive study.

An equivalence guideline of service S consists of two major ingredients; a complete canonical $\mathcal{B}$ substitute for S, and all its $\mathcal{B}$ covering situations. As studied in Chapter 5, the complete canonical $\mathcal{B}$ substitute for S is, by construction, structurally more liberal than any other $\mathcal{B}$ equivalent service of S because it encodes within its own structure all possible sequences and choices of events. Though relatively large, the complete canonical $\mathcal{B}$ substitute involves all possible situations that can influence any controller of S.

Intuitively, two situations of a given service are equivalent whenever they influence a partner in a similar way. We employed an equivalence relation to partition the relevant situations of the complete canonical $\mathcal{B}$ substitute for S to calculate the second ingredient of the equivalence guidelines. The partition allows us to identify all states which contribute to the situations in each subset of the partition, called $\mathcal{B}$ covering situations.

Service T matches with a $\mathcal{B}$ equivalence guideline of S whenever T refines the complete canonical $\mathcal{B}$ substitute for S under the refinement relation related to $\mathcal{B}$ and T matches with its $\mathcal{B}$ covering situations. We have proven that the set of all services that match with the equivalence guideline of S coincides with the set of all equivalent substitutes for S under $\mathcal{B}$. Therefore, we can employ a $\mathcal{B}$ equivalence guideline of service S and its matching relation to characterize the set of all equivalent substitution services of service S under behavioral compatibility criterion $\mathcal{B}$.

One useful application of a $\mathcal{B}$ equivalence guideline of a given service S is to synthesize a service that is equivalent to S under behavioral compatibility criterion $\mathcal{B}$. We can also apply the equivalence guideline to synthesize a service in a distinguished class of $\mathcal{B}$ equivalent services of S that satisfies specific properties, the so-called *public view* of S. We will discuss several applications of a $\mathcal{B}$ equivalence guideline of a given service in Chapter 7.

Part III.

Applications

7. Applications to Analysis and Synthesis for Service Substitution

In this section, we illustrate how to apply our results from Part II to solve analysis and synthesis problems related to service substitution. We discuss various scenarios of service substitution and present techniques that are related to the context. As established upon existing work on service substitution and correctness of services and their composition, we can combine our results with the related existing techniques to perform more sophisticated analysis and synthesis tasks on service substitution.

The chapter is organized as follows. Section 7.1 discusses techniques for analyzing service inclusion. Section 7.2 discusses techniques for analyzing service equivalence. Section 7.3 presents different techniques for synthesizing a substitute for a service under various scenarios. Section 7.4 proposes techniques for correcting a non-substitute of a service. Finally, Section 7.5 concludes the chapter.

7.1. Checking Service Inclusion

One of the problem class as investigated in this thesis addresses the correctness of service substitution by *verification*. Given services S and T, a service designer wants to know how to systematically *decide* whether or not service T is a *correct* substitute for service S ?

In this section, we present four different procedures to check whether service T is a substitute for service S under $\mathcal{B}$ inclusion for $\mathcal{B}$ is either deadlock freedom ($\mathcal{B} = df_k$) or responsiveness (i. e., $\mathcal{B} = rp_k$). The first procedure is inherited from the literature [Stahl, Massuthe, and Bretschneider, 2009] for checking deadlock freedom inclusion. The other three procedures are realized from our results from Part II in this thesis.

7.1.1. Comparing Operating Guidelines of Services

Though the set of $\mathcal{B}$-controllers of a given service S is possibly infinite, the set of all $\mathcal{B}$-controllers of S can be represented by a $\mathcal{B}$ operating guideline of S. [Stahl, Massuthe, and Bretschneider, 2009] have proposed a procedure for deciding whether service T is a substitute for service S under deadlock freedom inclusion (called *Accordance* in Stahl, Massuthe, and Bretschneider [2009]) by comparing a deadlock-free operating guideline of S and a deadlock-free operating guideline of T. We generalize this procedure for both deadlock freedom inclusion and responsiveness inclusion and illustrate the procedure in Figure 7.1.

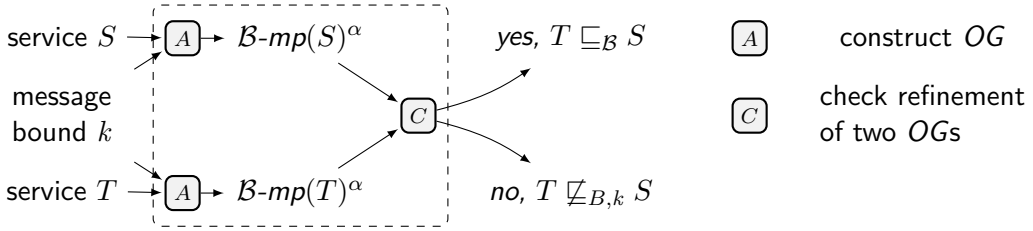


Figure 7.1.: Deciding service substitutability under $\mathcal{B}$ inclusion by comparing two operating guidelines of services, for $\mathcal{B} \in \{df_k, rp_k\}$.

The procedure (denoted by the dashed rectangle) first requires to construct a $\mathcal{B}$ operating guideline for both services S and T (denoted by process rectangle A) for a given message bound $k \in \mathbb{N}$ on each message channel. Then, the procedure uses the two operating guidelines to analyze whether every service described by the operating guideline of S is also described by the operating guideline of T (denoted by process rectangle C).

– For deadlock freedom ($\mathcal{B} = df_k$), process A constructs the two k-deadlock-free operating guidelines $mp_{df,k}(S)^\varphi$ from service S and $mp_{df,k}(T)^\varphi$ from service T (cf. Section 3.3.2). Process C involves two tasks; firstly checking if $mp_{df,k}(T)$ strongly simulates $mp_{df,k}(S)$ (that is, $mp_{df,k}(T)$ strongly simulates every k-deadlock-free controller of S) and checking if, for every simulated pair $[q_{mT}, q_{mS}]$ of state q_{mS} of $mp_{df,k}(S)$ and q_{mT}

of $mp_{df,k}(T)$, every deadlock-free choice at q_{mS} is also a deadlock-free choice at q_{mT}, i. e., $\varphi(q_{mS}) \subseteq \varphi(q_{mT})$. The correctness of this procedure is asserted by Corollary 3.55.

– For responsiveness ($\mathcal{B} = rp_k$), process A constructs the two k-responsive operating guidelines $mp_{rp,k}(S)^{\psi^*}$ from service S and $mp_{rp,k}(T)^{\psi^*}$ from service T (cf. Section 3.3.3). Process C involves two tasks; first checking if $mp_{rp,k}(T)$ strongly simulates $mp_{rp,k}(S)$ (that is, $mp_{rp,k}(T)$ strongly simulates every k-responsive controller of S) and checking if, for every simulated pair $[q_{mT}, q_{mS}]$ of state q_{mS} of $mp_{df,k}(S)$ and q_{mT} of $mp_{df,k}(T)$, every responsive choice at q_{mS} is also a responsive choice at q_{mT}, i. e., $\psi^*(q_{mS}) \subseteq \psi^*(q_{mT})$. The correctness of this procedure is asserted by Corollary 3.73.

To decide service inclusion of two services by checking their operating guidelines, we employ a number of open source software tools from *service-technology.org*[1].

In order to apply the procedure described in this section and in this thesis, we first translate WS-BPEL processes into service automata. As we focus on the *behavior* aspect of WS-BPEL processes, we abstract from other aspects such as data, instantiation, and time. For this purpose, we use the compiler *BPEL2oWFN* [Lohmann, 2007] and *PnAPI* [Mennicke, Sura, Waltemath, Lohmann, Gierds, and Znamirowski, 2009].

For deadlock freedom inclusion, we apply the tool *Fiona* [Massuthe and Weinberg, 2008] to construct the two deadlock-free operating guidelines and analyze their refinement relation. *Fiona* is a tool that provides several analysis and synthesis algorithms for service behavior. The design goal of Fiona was to combine several analysis and synthesis algorithms for service behavior. It provides a set of data structures and algorithms for promoting reusability, and therefore, facilitates for fast integration of new algorithms.

For responsiveness inclusion, we apply the tool *Wendy* [Lohmann and Weinberg, 2010] to construct the two responsive operating guidelines. *Wendy* is a tool for deciding service controllability and for synthesizing partners of services. The design goal of Wendy was to deliver an efficient solution for service analysis and synthesis problem in a more compact single-purpose tool. For the purpose of comparing two operating guidelines, we require a separate tool in the family.

Recently, several efforts have been carried out to optimize the space-efficiency of operating guidelines [Kaschner et al., 2007, Gierds, 2008a, Lohmann and Wolf, 2009]. One alternative representation of operating guidelines is a representation where, instead of an annotation at each state (e. g., Boolean formula), only a few bits need to be stored for a state [Lohmann and Wolf, 2011b]. Such a representation economizes the storage of operating guidelines and yields efficiency gains in decision algorithms for service inclusion involving operating guidelines.

Along this line, the tool *Cosme* [Lehmann, 2011] is able to compare the refinement relations of the two operating guidelines by taking the bits operating guidelines format as its input. To compare the two operating guidelines generated by *Wendy*, we use the complier *WendyFormula2bits* [Sura, 2009b] to translate the operating guidelines generated from *Wendy* into the bits operating guidelines, and use *Cosme* to compare the two bits operating guidelines.

[1] illustrated in Figure 2.9 and available at `http://service-technology.org/tools`

7.1.2. Checking Composition with Canonical Controller

In this section, we present a decision procedure for deciding service inclusion under $\mathcal{B}$ by checking the behavioral compatibility criterion $\mathcal{B}$ of the composition with a canonical controller of a given service. We illustrate this procedure with Figure 7.2 and Figure 7.3.

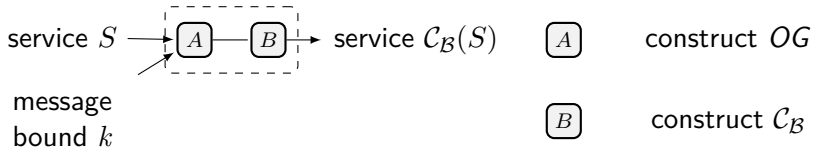


Figure 7.2.: Synthesizing a canonical controller $\mathcal{C}_{\mathcal{B}}(S)$ of a given service S in case $\mathcal{B} \in \{df_k, rp_k\}$.

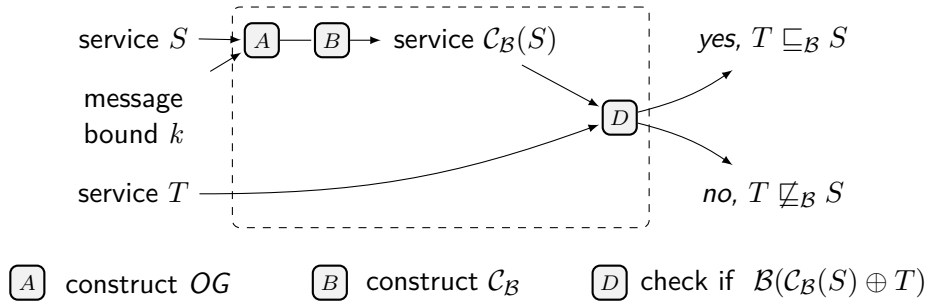


Figure 7.3.: Checking service substitutability under $\mathcal{B}$ inclusion using the composition with canonical controller, for $\mathcal{B} \in \{df_k, rp_k\}$.

The procedure (denoted by the dashed rectangle) first requires the construction of a canonical $\mathcal{B}$ controller of service S (denoted by process rectangles A and B) for a given message bound $k \in \mathbb{N}$ on each message channel. Then, the procedure verifies if the composition of service T and the constructed canonical $\mathcal{B}$ controller satisfies the behavioral compatibility criterion $\mathcal{B} \in \{df_k, rp_k\}$ (process D).

– In case of deadlock freedom ($\mathcal{B} = df_k$), processes A and process B for constructing a complete canonical k-deadlock-free controller of a given service are described by Definition 4.1. Alternatively, it is more desirable to construct a more compact version of a canonical k-deadlock-free controller as described by Definition 4.3. The correctness of this procedure is asserted by Theorem 5.6 and Theorem 5.11.

– In case of responsiveness ($\mathcal{B} = rp_k$), processes A and B for constructing a complete canonical k-deadlock-free controller of a given service are described by Definition 4.6. Similar to the case of $\mathcal{B} = df_k$, it is more desirable to construct a more compact version of a canonical k-deadlock-free controller as described by Definition 4.8. The correctness of this procedure is asserted by Theorem 5.6 and Theorem 5.17.

Given a service automaton model of S, we use the tool *Fiona* to construct a deadlock-free operating guideline for S and the tool *Wendy* to construct a responsive operating guideline for S (process A). The synthesis algorithm (process B) for two variants of canonical k-deadlock-free controller has been implemented in the tool *Maxis* [Parnjai, 2011c] under the course of this thesis. *Maxis* is a tool for synthesizing a maximal controller of a service from the input operating guideline. The synthesis of a maximal controller by *Maxis* is parameterized by either a complete or a compact form of a maximal controller. The verification of compatibility of service composition (process D) can be performed by the general-purpose Petri net model checking tool *LoLA* [Schmidt, 2000, Wolf, 2007b]. *LoLA* is a model checking tool which implements several state space reduction techniques.

In comparison to the procedure described in Section 7.1.1, the procedure shown in Figure 7.3 improves on deciding service inclusion for both deadlock freedom and responsiveness inclusion of services. This decision procedure requires the construction of only one canonical controller (process B) from a given service, compared to computing operating guidelines for both services S and T. As constructing a canonical controller can be regarded as a by-product of constructing an operating guideline, the construction of the canonical controller (process B) can be integrated into the construction of an operating guideline (process A). Obviously, the integrated process is required only once for service S but not for service T. As a result, the decision procedure (process D) is less expensive than computing the refinement relation of the two operating guidelines (process C). To optimize the decision procedure, we construct a compact canonical controller from a given service, instead of constructing its complete canonical controller. This definitely reduces the size of the state spaces needed for verifying the compatibility property of service composition.

7.1.3. Matching Operating Guidelines of Canonical Controller

In this section, we present a procedure for deciding service substitutability under $\mathcal{B}$ inclusion using an operating guideline of canonical controllers of services. This decision procedure checks for a matching relation with an operating guideline of a canonical controller of a service. We illustrate this decision procedure in Figure 7.4.

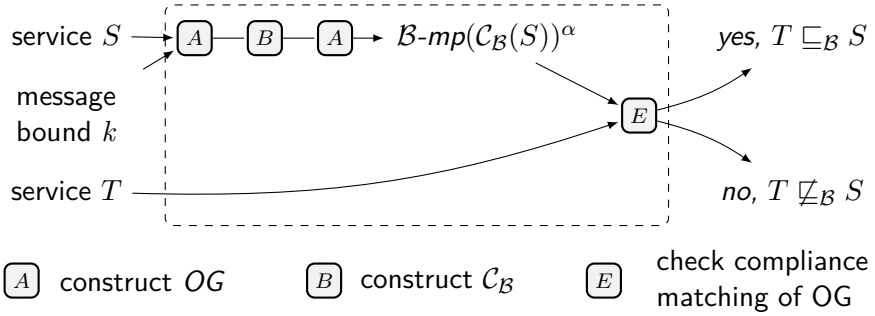


Figure 7.4.: Checking service substitutability under $\mathcal{B}$ inclusion by matching with an operating guideline of one service, for $\mathcal{B} \in \{df_k, rp_k\}$.

The procedure (denoted by the dashed rectangle) first requires the construction of a canonical $\mathcal{B}$ controller of service S (denoted by process rectangles A and B) for a given message bound $k \in \mathbb{N}$ on each message channel. Next, it requires to construct a $\mathcal{B}$ operating guideline of the constructed canonical controller (denoted by process rectangle A). Then, it checks whether or not service T is described by the constructed operating guideline (denoted by process rectangle E).

– In case of deadlock freedom ($\mathcal{B} = df_k$), processes A and B for constructing a canonical k-deadlock-free controller of a given service are described by Definition 4.1. Then, process A constructs a deadlock-free operating guideline from the constructed canonical controller. Process E involves checking if service T is described by the deadlock-free operating guideline of the canonical controller using strong compliance relation Definition 3.15. The correctness of this procedure is asserted by Theorem 5.7, Lemma 3.51, and Theorem 5.11.

– In case of responsiveness ($\mathcal{B} = rp_k$), processes A and B for constructing a canonical k-responsive controller of a given service are described by Definition 4.6. Then, process A constructs a responsive operating guideline from the constructed controller. Process E involves checking if service T is described by the operating guideline of the canonical controller using the structural compliance relation from Definition 3.17. The correctness of this procedure is asserted by Theorem 5.7, Lemma 3.69, and Theorem 5.17.

Given a service automaton model of service S, we can employ our open source software tool *Evans* [Parnjai, 2011b] developed under the course of this thesis. *Evans* [Parnjai, 2011b] synthesizes the desired operating guideline of a canonical controller of service S by pipelining a given service S through the other tools. First it invokes either *Fiona* or *Wendy* to generate an operating guideline for S (process A), then *Maxis* to generate a canonical controller of S from the operating guideline (process B), and finally either *Fiona* or *Wendy* again to generate an operating guideline of the canonical controller. By doing so, *Evans* delivers an operating guideline of a canonical substitute for a given service as its output. Similarly to the procedure described in Section 7.1.2, it is possible to construct a compact canonical controller by parameterizing *Maxis*. So far, the implementation of the checking algorithm (process E) is still under development, but will be integrated into the tool *Evans*.

In comparison to the method of checking the composition with the canonical controller described in Section 7.1.2, this procedure requires one additional step for computing the operating guideline of the canonical controller. Nevertheless, it is possible to construct a more compact version of a canonical $\mathcal{B}$-controller rather than to construct a complete canonical $\mathcal{B}$-controller of a given service. The compact canonical $\mathcal{B}$-controller of a given service certainly speeds up the computation of operating guidelines. Importantly, the deciding procedure described by process E is less expensive than process D (described in Section 7.1.2). This is because process E checks for a matching relation between service T and the computed operating guideline. However, process E does not require to explore state spaces of the composition in order to verify if the composition satisfies the given behavioral compatibility criterion $\mathcal{B}$, as described by process D in Section 7.1.2.

7.1.4. Checking Refinement of Canonical Substitute

In the section, we present a procedure for deciding service substitutability under $\mathcal{B}$ inclusion using the refinement relation on a canonical substitute for services. This decision procedure compares the respective failures semantics of a given service with its canonical substitute. We illustrate this procedure in Figure 7.5 and Figure 7.6.

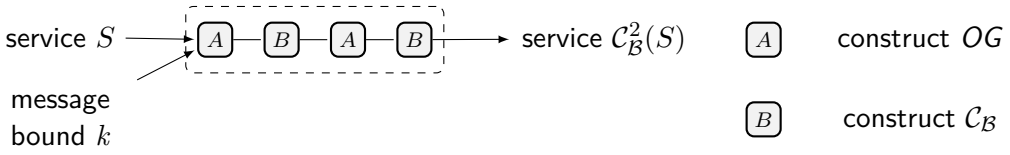


Figure 7.5.: Synthesizing a canonical substitute $\mathcal{C}^2_{\mathcal{B}}(S)$ for a given service S in case $\mathcal{B} \in \{df_k, rp_k\}$.

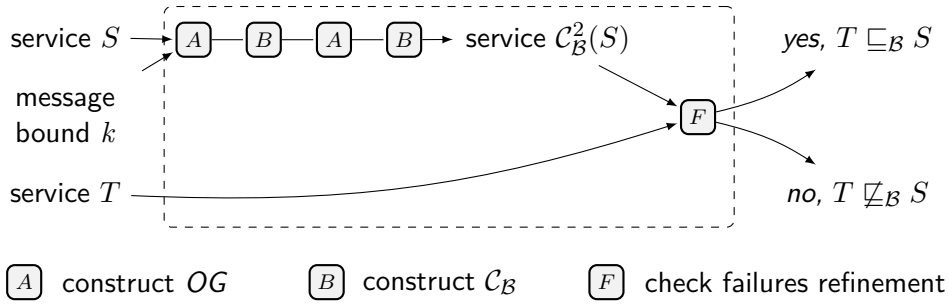


Figure 7.6.: Checking service substitutability under $\mathcal{B}$ inclusion using refinement relation between a given service and its canonical $\mathcal{B}$ substitute, for $\mathcal{B} \in \{df_k, rp_k\}$.

The procedure (denoted by the dashed rectangle) first requires the construction of a canonical $\mathcal{B}$ substitute for a given service S (denoted by process rectangles A, B, A, and B) for a given message bound $k \in \mathbb{N}$ on each message channel. Then, it checks whether service T refines the constructed canonical $\mathcal{B}$ substitute for service S under the respective failures (denoted by process rectangle F).

– In case of deadlock freedom ($\mathcal{B} = df_k$), the construction of a canonical k-deadlock-free substitute for service S is described by Definition 5.30 (the chained process A, B, A and B). Then, the procedure checks whether service T refines the constructed canonical k-deadlock-free substitute under stable failures (process F). The correctness of this procedure is asserted by Theorem 5.32.

– In case of responsiveness ($\mathcal{B} = rp_k$), the four-chained process A, B, A and B to construct a canonical k-responsive substitute for service S is described by Definition 5.34. Then, the procedure checks whether service T refines the constructed canonical k-responsive substitute under responsive failures (process F). The correctness of this procedure is asserted by Theorem 5.36.

The construction algorithm of the canonical substitutes for services (the chained process A, B, A and B) has been implemented in the open source software tool *Evans* [Parnjai, 2011b] under the course of this thesis. *Evans* pipelines a given service through the other tools; either *Fiona* or *Wendy*, and *Maxis*. By doing so, *Evans* delivers a canonical substitute for a given service as its output. The implementation of the checking algorithm (process E) is currently under development and will be integrated into the tool *Evans*.

This procedure is comparable to the matching of operating guideline of canonical controllers described in Section 7.1.3. Though this procedure requires the construction of one canonical substitute (the last process B) from the operating guideline (the chained process A, B, and A), the construction of a canonical substitute is a by-product of constructing an operating guideline, and therefore, can be integrated into the construction of an operating guideline. Furthermore, it is possible to construct a more compact version of a canonical $\mathcal{B}$-substitute rather than to construct a complete canonical $\mathcal{B}$-substitute for a given service. The decision procedure (process F) is to find refinement relation between two services and it is comparable to process E described in Section 7.1.3.

In Section 7.3, we will illustrate that the canonical substitute for a service is more suitable for solving the synthesis problem rather than the analysis problem. For instance, we can employ the canonical substitute for a given service to synthesize a substitute service with additional behavioral constraints.

7.2. Checking Service Equivalence

In this section, we present two techniques to check whether a given service T is a substitute for a given service S under $\mathcal{B}$ equivalence for $\mathcal{B}$ is either deadlock freedom ($\mathcal{B} = df_k$) or responsiveness (i. e., $\mathcal{B} = rp_k$). Similar to Section 7.1, the first procedure described in Section 7.1.1 is inherited from the literature [Stahl, Massuthe, and Bretschneider, 2009] to decide deadlock freedom preorder. Another procedure is realized from our results from Part II of this thesis.

7.2.1. Comparing Operating Guidelines of Services

Similar to Section 7.1.1, deciding whether or not a service T can substitute a service S under $\mathcal{B}$ equivalence can also be done by comparing the $\mathcal{B}$ operating guidelines of the two services. Stahl, Massuthe, and Bretschneider [2009] have proposed a procedure for deciding whether service T can substitute service S under deadlock freedom equivalence (called *Accordance equivalence* in Stahl et al. [2009]) by comparing a deadlock-free operating guideline of S and a deadlock-free operating guideline of T. We generalize this procedure for both deadlock freedom equivalence and responsiveness equivalence and illustrate this procedure with Figure 7.7.

The procedure (denoted by the dashed rectangle) first requires to construct a $\mathcal{B}$ operating guideline for both services S and T (denoted by process rectangle A) for a given message bound $k \in \mathbb{N}$ on each message channel. Then, the procedure uses the two operating guidelines to analyze whether they are equivalent (denoted by process rectangle G).

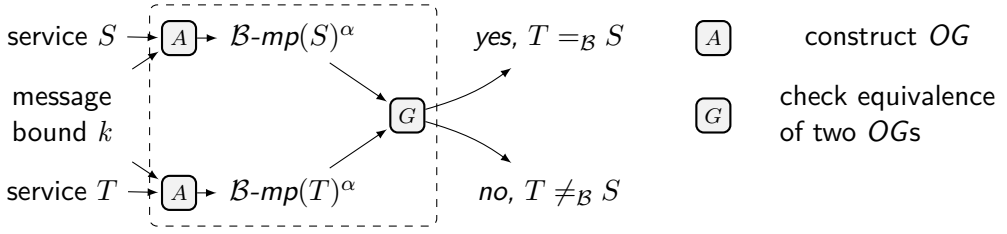


Figure 7.7.: Checking substitutability of services under $\mathcal{B}$ equivalence by comparing operating guidelines of two services, for $\mathcal{B} \in \{df_k, rp_k\}$.

– In case of deadlock freedom ($\mathcal{B} = df_k$), process A constructs the two k-deadlock-free operating guidelines $mp_{df,k}(S)^{\varphi}$ from service S and $mp_{df,k}(T)^{\varphi}$ from service T (cf. Section 3.3.2). Then, process G involves two tasks. Firstly, checking if the two operating guidelines are bisimilar. Secondly, checking if, for every pair $[q_{mT}, q_{mS}]$ of bisimilar states q_{mS} of $mp_{df,k}(S)$ and q_{mT} of $mp_{df,k}(T)$, whether the set of deadlock-free choices at q_{mS} and the set of deadlock-free choices at q_{mT} are the same, i. e., $\varphi(q_{mS}) = \varphi(q_{mT})$. The correctness of this procedure is asserted by Corollary 3.56.

– In case of responsiveness ($\mathcal{B} = rp_k$), process A constructs the two k-responsive operating guidelines $mp_{rp,k}(S)^{\psi^*}$ from service S and $mp_{rp,k}(T)^{\psi^*}$ from service T (cf. Section 3.3.3). Then, process G involves two tasks. Firstly, checking if the two operating guidelines are bisimilar. Secondly, checking if, for every pair $[q_{mT}, q_{mS}]$ of bisimilar states q_{mS} of $mp_{rp,k}(S)$ and q_{mT} of $mp_{rp,k}(T)$, whether the set of valid responsive choices at q_{mS} and the set of valid responsive choices at q_{mT} are the same, i. e., $\psi(q_{mS}) = \psi(q_{mT})$. The correctness of this procedure is asserted by Corollary 3.74.

This procedure checks for an equivalence relation of the $\mathcal{B}$-operating guidelines of two given services. As a $\mathcal{B}$-operating guideline of a given service is a finite representation of all its $\mathcal{B}$ controllers, it is also possible to check for an equivalence relation on other representations of services than the operating guideline, such as checking for an equivalence relation between canonical $\mathcal{B}$ controllers or canonical $\mathcal{B}$ substitutes for two services. Obviously, such procedures can never outperform the checking of the two operating guidelines of two services, because those services are synthesized on top of operating guidelines and the equivalence check is a symmetric procedure that must be performed in two directions.

Similar to the procedure described in Section 7.1.1, we use similar tools to check the equivalence relation of two constructed operating guidelines.

7.2.2. Matching Equivalence Guidelines of Services

In the section, we present a procedure for deciding service substitutability under $\mathcal{B}$ equivalence using an equivalence guidelines of a service. This decision procedure check whether or not one service is described by the equivalence guidelines of another service We illustrate this procedure in Figure 7.8.

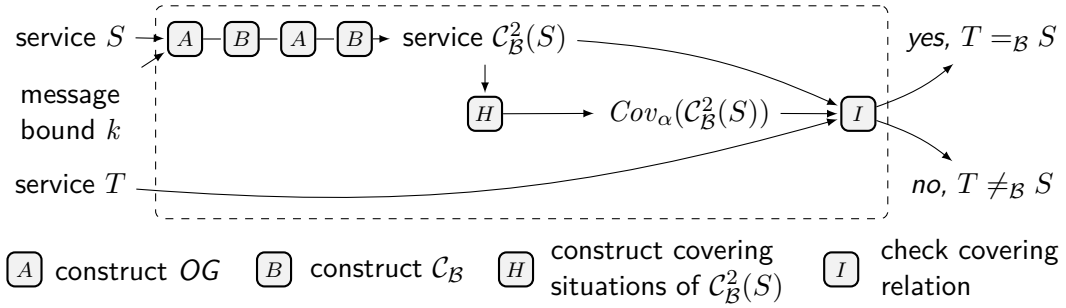


Figure 7.8.: Checking service substitutability under $\mathcal{B}$ equivalence by matching with an equivalence guideline of a service, for $\mathcal{B} \in \{df_k, rp_k\}$.

The procedure (denoted by the dashed rectangle) first requires the construction of a canonical $\mathcal{B}$ substitute for a given service S (denoted by process rectangles A, B, A, and B) for a given message bound $k \in \mathbb{N}$ on each message channel. Then, it constructs from the canonical $\mathcal{B}$ substitute its covering situations (denoted by process rectangle H). To decide if service T is a substitute for service S under $\mathcal{B}$ equivalence, the procedure checks whether service T is described by the covering situations of the canonical $\mathcal{B}$ substitute for service S (denoted by process rectangle I).

– In case of deadlock freedom ($\mathcal{B} = df_k$), the construction of a complete canonical k-deadlock-free substitute for service S is described by Definition 5.30 (the chained process A, B, A, and B). The construction of the deadlock-free covering situations of a complete canonical k-deadlock-free substitute for service S is described by Definition 6.6 (process H). Checking whether service T is a substitute for service S under deadlock freedom equivalence (process I) involves two steps; first, checking if service T refines the complete canonical k-deadlock-free substitute under stable failures, and second, checking if service T matches with the deadlock-free covering situations of the complete canonical k-deadlock-frees substitute. The correctness of this procedure is asserted by Theorem 6.14.

– In case of responsiveness ($\mathcal{B} = rp_k$), the four-chained process A, B, A and B to construct a canonical k-responsive substitute for service S is described by Definition 5.34. The construction of the responsive covering situations of a complete canonical k-responsive substitute for service S is described by Definition 6.20 (process H). Checking whether service T is a substitute for service S under responsiveness equivalence (process I) involves two steps; first, checking if service T refines the complete canonical k-responsive substitute under responsive failures, and second, checking if service T matches with the responsive covering situations of the complete canonical k-responsive substitute. The correctness of this procedure is asserted by Theorem 6.28.

This procedure checks for a matching relation with a $\mathcal{B}$-equivalence guideline of service S. In comparison to the procedure described in Section 7.2.1, this procedure requires

more pre-processing steps to computer a complete canonical $\mathcal{B}$ substitute for service S (the chained process A, B, A, and B) and its covering situations (process H). Process H can be expensive as it requires to explore the state space of the composition between a canonical controller of S and a canonical substitute for S. Nevertheless, the decision procedure (process I) does not require the construction of an operating guideline of T, but requires to check whether service T can cover the covering situations described by the equivalence guideline. This check can be done by exploring the state space of the composition between service T and the most permissive controller of S. Clearly, this procedure does not outperform the checking of the operating guidelines of two services described in Section 7.2.1. In Section 7.3 and Section 7.4, we will illustrate that the equivalence guidelines are more suitable for solving synthesis problem rather than analysis problem. For instance, we can employ the equivalence guidelines to synthesize a minimal public view for a given service.

The implementation of an equivalence guideline is currently under development and will be integrated into the open source software tool *Evans*.

7.3. Synthesizing Substitutes for Services

Whenever the behavior of service S changes due to various reasons, e.g., changes in regulations or the operational behavior of services, an incremental modification of an existing protocol requires a construction of a new service T without the need of redefining T from scratch. Yet, designing a new service T that is a substitute for service S is a time-consuming and error-prone task, typically based on trial-and-error methods.

In this section, we present a number of synthesis techniques that provide support for optimizing the time and effort to design such a service in various scenarios. All presented techniques are derived from the results described in Part II of this thesis in combination with related existing techniques from the literature (e. g., Lohmann et al. [2007b], Stahl et al. [2009], Stahl and Wolf [2008]).

To apply the synthesis techniques presented in this thesis on industrial services, we apply the compilers and tools in the family *service-technology.org*[2]. First, we translate WS-BPEL processes into our formal model using the compiler *BPEL2oWFN* [Lohmann, 2007] and *PnAPI* [Mennicke et al., 2009]. Then, we apply one of our synthesis procedures which produces a service automaton model as an output. The output service automaton can be translated into an abstract *WS-BPEL* process, using the compiler *PnAPI* and *oWFN2BPEL* [Lohmann and Kleine, 2008]. To this respect, the synthesized service is *communication skeleton* which need to be refined further manually. Therefore, a number of tools in the family provides support for closing the loop between our formal model and the real-life services.

The remainder of this section is organized as follows. Section 7.3.1 discusses a technique to synthesize a substitute for a service by means of transformation. Section 7.3.2 describes a method to avoid design flaws by enforcing the correctness of substitution by construction, and discusses several techniques to construct a substitute for a service that satisfies

[2]illustrated in Figure 2.9 and available at `http://service-technology.org/tools`

specific constraints. Along the line of enforcing correctness by construction, Section 7.3.3 discusses a technique to construct three variants of public views for a given service. In a situation where preserving every controller of a given service is too restrictive, Section 7.3.4 describes a technique to synthesize a substitute for a given service which only preserves all selected controllers. In a different situation where all controllers of multiple services must be preserved, Section 7.3.4 describes a technique to synthesize a substitute for all services that are given.

7.3.1. Synthesizing Substitutes for Services by Transformation

In practice, one promising method to synthesize a service T that is a substitute for a given service S under a given compatibility criterion is by *transformation*. Given service S, we can apply a rule to transform service S into service T, which guarantees that T is a substitute for service S in every possible context relevant to a given substitutability criterion.

In this thesis, we present ten property-preserving transformation rules defined in the style of the Murata rules [Murata, 1989]. Some of these rules can be seen as inherited and extended from the literature, e. g., from van der Aalst and Basten [2002], van der Aalst et al. [2008], König et al. [2008], van der Aalst et al. [2009]. We illustrate (cf. Section 4.5),in Chapter Section 4) that each transformation rules preserves at least one of the substitution criteria studied in this thesis. Some transformation rules preserve stable failures refinement (cf. Section 4.5.1) or responsive failures refinement (cf. Section 4.5.2) or both (cf. Section 4.5.3). Some of the rules guarantee to preserve the property in both directions. As the stable failures refinement is finer than deadlock freedom inclusion (cf. Section 4.3.3), the rule that preserves either stable failures refinement or equivalence, by implication, also preserves deadlock freedom inclusion or equivalence respectively. Similarly to the rules that preserves responsive failures refinement, as responsive failures refinement is finer than responsiveness inclusion (cf. Section 4.4.3).

Synthesizing a substitute for a service by means of transformation is useful for designing many real-life service models. Nevertheless, most of the transformation rules presented in the literature and in this thesis are limited, as many rules are restricted to add or remove either internal τ transitions or choices of services. Though some extensions regarding behavioral compatibility (e. g., in König et al. [2008] and van der Aalst et al. [2009]) are motivated by the semantics of a service instead of its syntax, and the rules allows reordering of the communication events by restricting the set of permitted executable completions, the set of these rules are still incomplete. This means, it is not possible to construct every substitute for a given S by applying only existing transformation rules.

To cover as much substitution as possible for a given service S, we propose to combine the method of rule-based transformation with the synthesis of a complete canonical substitute for service S (cf. Section 5.2.2). Given service S, we first synthesis a complete canonical substitute for service S. The synthesized complete canonical substitute for service S represents all substitutes for service S under $\mathcal{B}$ inclusion and equivalence. A substitute for service S can be synthesized from its complete canonical substitute by applying existing transformation rules. Though the size of the complete canonical

substitute for service S can be relatively larger than service S itself, the canonical substitute can be seen as a skeleton that contains all possible communicating behavior of a substitute for service S. Then, we further refine the canonical substitute incrementally by applying the property-preserving transformation rules, e. g., to removed unwanted behavior from the canonical substitute, before filling in the details of each internal event. For illustrating examples of this method, we refer to Figure 5.4 from Section 5.1.2 and from Figure 5.5 from Section 5.1.3.

7.3.2. Synthesizing Substitutes for Services by Construction

An alternative method to enforce correctness of service substitution is *by construction*. With this method, a service model is constructed from a specification and the correctness of substitution follows from the correctness of the construction algorithm. Such construction aims to avoid design flaws at the early phase of development. Incrementally the synthesized service model can be refined further either manually or automatically towards its implementation. Suppose a service designer wants to synthesize a substitute service T for a given service S such that T also satisfies additional behavioral constraints on service S. For instance, a service designer want to characterize a customer service of a travel agency whom never abort after sending his/her first request, or characterize an online shop service that accepts a credit card payment after receiving an order.

To provide support for a service designer in this sense, we can combine our results from Part II with related existing works based on the operating guidelines approach, such as to characterize partners that enforce or exclude certain activities [Lohmann et al., 2007b] or to characterize partners that cover certain activities in less restricted manner [Stahl and Wolf, 2008], or to characterize services that must not exclude selected partners [Stahl et al., 2009]. To apply these techniques, we construct an operating guideline that represents all substitutes for a given service under $\mathcal{B}$ inclusion for behavioral compatibility criterion $\mathcal{B} \in \{df_k, rp_k\}$ of service composition. Given service S and compatibility criterion $\mathcal{B}$, we first construct a canonical $\mathcal{B}$ controller for service S (cf. Section 4.1), then construct a $\mathcal{B}$ operating guideline of the canonical controller. The constructed operating guideline is a finite representation of all substitutes for service S under $\mathcal{B}$ inclusion (cf. Section 5.2.1). To this respect, all related existing techniques for operating guidelines are also applicable to our results under our setting of service substitution.

With our results from Part II, we also improve related techniques for synthesizing substitutes for services that preserves selected partners, and for correcting a service behavior in a service choreography. Respectively, we discuss these techniques further in Section 7.3.4 and Section 7.4.1.

7.3.3. Synthesizing Public Views for Services

In the context of inter-organizational processes, the service-oriented approach enables an organization to develop a service without knowing its partner in advance. To find a service partner, a service provider needs to publish information about the services it offers. Information about the published services will be stored in a service repository.

From the repository, a service requester discovers the compatible services that meet the requirements. Once this information is agreed with by a service requester, the dynamic binding of services from provider and requester takes place at runtime. In order to facilitate the process of service recovery, a service provider must provide sufficient details of the published services if offers. On the other hand, the organization policy may require to hide the irrelevant internal details of the published services from other organizations [van der Aalst and Weske, 2001, Leymann et al., 2002].

In many situations, it is also possible for an organization to cooperate with others before developing own services. Therefore, these organizations agree beforehand on a common *contract* [van der Aalst and Basten, 2002, van der Aalst and Weske, 2001, van der Aalst et al., 2008]. To enforce interoperability between organizations, a contract defines essential details of process interactions, which the participating organization must conform to. At the same time, a contract allows each organization to autonomously modify own process whenever needed.

A popular approach to tackle the problem in these two scenarios is to employ a *public view* for a service. Instead of publishing the service, a service provider may publish its public view that can act on behalf of the service. Yet, a pubic view for a service does not reveal its irrelevant internal details. To this respect, a public view for a service can also be regarded as a specification of service interaction that each organization must conform to. Each organization can refine its part of the specification into a *private view*, which may significantly differ from the public view.

In this thesis, we present the procedures to construct two variants of public views from a given service S under the behavioral compatibility criterion $\mathcal{B} \in \{df_k, rp_k\}$. Each public view for a given service S is a substitute for service S under $\mathcal{B}$ equivalence and can act on behalf of service S under a given substitution criterion. Depending the use case of a public view, each variant has it own advantages for solving the problem under different situations.

Synthesizing a Liberal Public View for a Service

In this section, we present a *liberal public view* for a service S. Intuitively, a public view for service S is *liberal* in the sense that it is a substitute for service S under $\mathcal{B}$ equivalence with a high degree of concurrency of asynchronously communicating events for $\mathcal{B} \in \{df_k, rp_k\}$.

Given a $\mathcal{B}$-controllable service S, a liberal public view for S can be constructed as a complete canonical substitute for service S (cf. Section 5.2.2). By construction, a liberal public view for service S encodes within its own structure all possible traces and communication choices with every $\mathcal{B}$-controller of S. To this respect, we can regard a most liberal public view of service S as a (behavioral) specification for service S. Then we can refine the specification further e. g., by means of transformation (cf. Section 7.3.1), to add or remove unwanted choices or internal τ events.

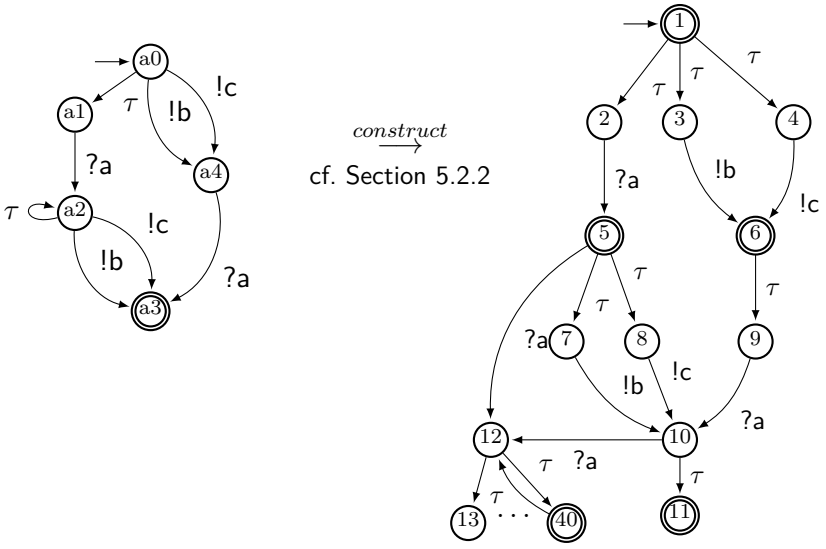
Example 7.1. For an illustration of a liberal public view for a service, we refer to Figure 5.8 and Example 5.33 in case of deadlock freedom ($\mathcal{B} = df_k$), and to Figure 5.10 and Example 5.37 in case of responsiveness ($\mathcal{B} = rp_k$). ◁

Given a service S, the algorithm generates a service that contains all possible traces and choices that can interact with every $\mathcal{B}$-controller of S. The constructed liberal public view can be relatively large in comparison to S. The size of the constructed liberal public view is indeed growing exponentially with respect to the size of a service (cf. Section 5.2.3).

To improve efficiency of a further operation on public views, it is an advantage to derive a smaller public view. For this purpose, we present an optimized version of a liberal public view as a compact canonical substitute for service S (cf. Section 5.2.2). By construction, an optimized liberal public view for service S encodes within its own structure all possible traces, and only canonical choices describing asynchronous communication with every $\mathcal{B}$-controller of S.

Example 7.2. For an illustration of an optimized liberal public view for a service, we refer to Figure 5.9 and Example 5.33 in case of deadlock freedom ($\mathcal{B} = df_k$), and to Figure 5.11 and Example 5.37 in case of responsiveness ($\mathcal{B} = rp_k$).

Another example of an optimized public view for services is illustrated in Figure 7.9 where an optimized liberal public view for service U_1 on the right hand side is constructed from U_1 as it compact canonical k-responsive substitute $\widehat{\mathcal{C}}^2_{rp,k}(U_1)$ for message bound $k = 1$ on each message channel. ◁



(a) U_1 (b) an optimized liberal public view of U_1

Figure 7.9.: Service U_1 and its liberal public view constructed as a compact canonical k-responsive substitute $\widehat{\mathcal{C}}^2_{rp,k}(U_1)$ for U_1 for message bound $k = 1$ on each message channel. Both services have the interface $I = \{a\}$ and $O = \{b, c, d\}$.

An optimized liberal public view is relatively small in comparison to the non-optimized liberal public view (cf. Section 5.2.3). Though, it is possibly larger than the service S,

as our algorithm systematically add internal τ options for all choices that are possible for a substitute for service S. Though it is possible to apply a $\mathcal{B}$-equivalence-preserving transformation rules to reduce $\widehat{\mathcal{C}}^2_{rp,k}(U_1)$ further, the applications of transformation rules are restricted and still not complete.

Our procedure for constructing public view is related to the procedure described by Wolf [2007a] and Laufer [2007]. Thereby, a public view for service S is constructed from the operating guideline of S by means of reversing the operating guideline. Given an operating guideline of service S, the procedure constructs as a mirror of this operating guideline and check if addition transitions must be inserted. Therefore, the size of the constructed public view is equal to the size of the operating guideline plus extra nodes resulting from checking a special case. As a result, the public view for service S constructed by this means is possibly larger than S itself, though it is relatively smaller than our liberal public view for service S. This related algorithm has been implemented by Laufer [2007] and integrated into the tool *Fiona*. Recent development on generating a public view from the bit representation of operating guideline has been implemented in the tool *Sayo* [Sura, 2009a].

In the course of this thesis, the construction algorithm of the two variants of liberal public views for services has been implemented in the open source software tool *Evans* [Parnjai, 2011b]. *Evans* pipelines a given service through the other tools *Wendy* and *Maxis*, and by doing so, delivers a liberal public view for a given service as its output.

In the next section, we will present an alternative construction procedure of a public view for constructing a *minimal* public view for a given service, containing a smallest possible number of states and smallest possible degree of concurrency of events asynchronously communicating with every controller S.

Synthesizing a Minimal Public View for a Service

In this section, we present a *minimal public view* for service S. In contrast to a liberal public view, a minimal public view for service S is a substitute for service S under $\mathcal{B}$ equivalence with a smaller possible degree of concurrency of asynchronously communicating events for $\mathcal{B} \in \{df_k, rp_k\}$.

Given a $\mathcal{B}$-controllable service S, a minimal public view for S can be constructed from a given service. The construction of a minimal public view requires similar information needed for constructing an equivalence guideline for service S (cf. Section 6.2.4 in case $\mathcal{B} = df_k$ and Section 6.3.4 in case $\mathcal{B} = rp_k$). The procedure aims to construct a substitute for service S that is smallest possible. The construction of a minimal public view for service S is described as follows.

Given a $\mathcal{B}$-controllable service S, the procedure first constructs a liberal public view for S as its compact canonical substitute $\widehat{\mathcal{C}}^2_{\mathcal{B}}(S)$. Alternatively, we can construct the complete canonical substitute $\mathcal{C}^2_{\mathcal{B}}(S)$ for S, instead of the compact canonical substitute $\widehat{\mathcal{C}}^2_{\mathcal{B}}(S)$ for S. Nevertheless, the complete canonical substitute $\mathcal{C}^2_{\mathcal{B}}(S)$ encodes within its own structure *all* possible communication traces and choices with every $\mathcal{B}$-controller of S, whereas the compact substitute $\widehat{\mathcal{C}}^2_{\mathcal{B}}(S)$ encodes only *essential* possible communication traces and choices. For the purpose of synthesizing a substitute for S under $\mathcal{B}$ equivalence that is

Algorithm 1: construction of a minimal public view

input : (1) $\mathcal{B}$-controllable service S and (2) message bound k
output : a minimal public view $PV(S)$
begin
  compute a compact canonical substitute $\widehat{\mathcal{C}}^2_{\mathcal{B}}(S)$ of S ;
  compute a most permissive controller $\mathcal{B}$-$mp(\widehat{\mathcal{C}}^2_{\mathcal{B}}(S))$ of $\widehat{\mathcal{C}}^2_{\mathcal{B}}(S)$;
  for *each state* q_m *of* $\mathcal{B}$-$mp(\widehat{\mathcal{C}}^2_{\mathcal{B}}(S))$ **do**
    compute a transition system TS of situations of $\widehat{\mathcal{C}}^2_{\mathcal{B}}(S)$ associated with state q_m ;
    compute the partition $E = \xi_{\widehat{\Sigma}(q_m)}\widehat{\mathcal{K}}(q_m)$ at q_m ;
    for *each* e *in* E **do**
      $[q_S, \mathcal{M}_S] = select(e)$;
      if q_S *is not marked* **then** $mark([q_S,\mathcal{M}_S,], TS)$;
  derive $PV(S)$ from $\widehat{\mathcal{C}}^2_{\mathcal{B}}(S)$ by removing all states that are not marked including their adjacent transitions ;
  apply transformation rules $F(1)$, $R(1)$, and $R(2)$ wherever possible on $PV(S)$ to remove τ transitions ;

minimal, the complete canonical substitute is more than necessary, as it is relatively larger and requires more computation (cf. Section 5.2.2 and Figure 4.1.3). Though the construction of the complete canonical substitute $\mathcal{C}^2_{\mathcal{B}}(S)$ for S is necessary for constructing the equivalence guideline for S, as an equivalence guideline for service S characterizes *all* substitutes for S under $\mathcal{B}$ equivalence (cf. Section 6.2.4 and Section 6.3.4).

Next, the procedure computes a most permissive controller $\mathcal{B}$-$mp(\widehat{\mathcal{C}}^2_{\mathcal{B}}(S))$ of $\widehat{\mathcal{C}}^2_{\mathcal{B}}(S)$. For each state q_m of the most permissive controller $\mathcal{B}$-$mp(\widehat{\mathcal{C}}^2_{\mathcal{B}}(S))$, the algorithm computes a transition system TS of situations of $\widehat{\mathcal{C}}^2_{\mathcal{B}}(S)$ associated with state q_m and then partitions the relevant situations $\widehat{\mathcal{K}}(q_m)$ of the complete substitute $\widehat{\mathcal{C}}^2_{\mathcal{B}}(S)$ using the equivalence relations of situations with respect to the set of canonical events $\widehat{\Sigma}(q_m)$. The calculated partition is denoted by $E = \xi_{\widehat{\Sigma}(q_m)}\widehat{\mathcal{K}}(q_m)$.

Then, for each $e \in E$, the situation $[q_S, \mathcal{M}_S]$ of $\widehat{\mathcal{C}}^2_{\mathcal{B}}(S)$ is selected from the set e using the following preference:

1. choose a situation $[q_S, \mathcal{M}_S]$ in which state q_S of $\widehat{\mathcal{C}}^2_{\mathcal{B}}(S)$ is already marked, otherwise,
2. choose a situation $[q_S, \mathcal{M}_S]$ in which state q_S of $\widehat{\mathcal{C}}^2_{\mathcal{B}}(S)$ offers a smallest possible choices among all other states associated with a situation in the same set e. In case there are more than one state that offers a smallest possible choice, choose one state arbitrarily.

In case state q_S associated with the selected situation $[q_S, \mathcal{M}_S]$ from e is not marked, the algorithm marks state q_S and other states of $\widehat{\mathcal{C}}^2_{\mathcal{B}}(S)$ by selecting a path in a transition TS that reaches the situation $[q_S, \mathcal{M}_S]$. The selection criterion is described as:

– for each step along the path, choose a predecessor situation $[q, \mathcal{M}]$ in which q is already

marked.
- in case none of the states of all predecessor situations is marked, choose a predecessor situation $[q, \mathcal{M}]$ in which state q offers a smallest possible choice among all other states associated with other predecessor situations. In case there are more than one state that offer a smallest possible choices, choose one state arbitrarily.

At this step, the algorithm marks every state q of $\widehat{\mathcal{C}}^2_{\mathcal{B}}(S)$ that is associated with all situations along the selected path.

Then, the public view $PV(S)$ for service S is derived from the complete substitute $\widehat{\mathcal{C}}^2_{\mathcal{B}}(S)$ by removing from $\widehat{\mathcal{C}}^2_{\mathcal{B}}(S)$ all states that are not marked including their adjacent transitions. At this stage, the public view $PV(S)$ for S preserves a choice of the compact canonical substitute $\widehat{\mathcal{C}}^2_{\mathcal{B}}(S)$ for S along each chosen path. This means, it refines the compact canonical substitute $\widehat{\mathcal{C}}^2_{\mathcal{B}}(S)$ for service S under respective failures, therefore, it is indeed a substitute for S (cf. Theorem 5.32 in case $\mathcal{B} = df_k$ and Theorem 5.36 in case $\mathcal{B} = rp_k$) Nevertheless, the public view $PV(S)$ at this stage contains internal τ events along each path due to the structure inherited from $\widehat{\mathcal{C}}^2_{\mathcal{B}}(S)$.

For the final step, the algorithm applies transformation rules $F(1)$ (cf. Definition 4.68), $R(1)$ (cf. Definition 4.83), and $R(2)$ (cf. Definition 4.85) wherever possible on $PV(S)$ to remove τ transitions as well as unnecessary final states. The correctness of the rules (cf. Lemma 4.69 and Lemma 4.84) ensures that the transformed $PV(S)$ is a substitute for service S under $\mathcal{B}$ equivalence for $\mathcal{B} \in \{df_k, rp_k\}$.

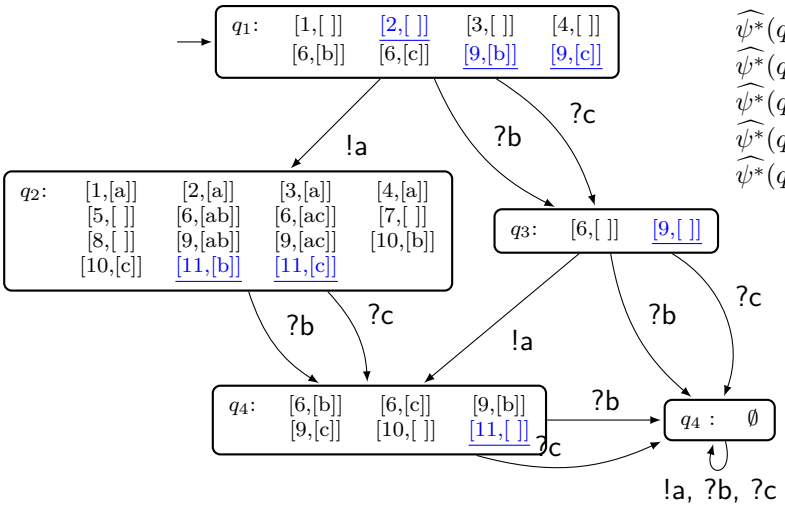
Clearly, the generated minimal public view from Algorithm 1 is not unique. Our criterion for selecting a path guarantees a smallest possible enabled choices along the path that are associated within the transition associated with the same state of most permissive controller. Though, it does not search an optimal path among all possible solutions in the other sets of covering situations. In addition, in case there is no local optimal solution found, a path is chosen arbitrarily (See Figure 7.10, for instance).

Example 7.3. Consider message bound $k = 1$ and k-responsiveness of service composition ($\mathcal{B} = rp_k$). We recall service S_1 from Figure 4.1, and its complete canonical k-responsive substitute from Figure 5.11. An output of Algorithm 1 on S_1 for $k = 1$ is either service S_1 itself or service S_2 from Figure 4.1.

Figure 7.11 illustrates three minimal public views for service U_1 from Figure 7.9. Each public view has the same set of interfaces as U_1 ($I = \{a\}$ and $O = \{b, c\}$) and is generated by applying Algorithm 1 on service U_1 for $k = 1$. Parts of the calculation are illustrated in Figure 7.10. Every service that is a substitute for service U_1 must be able to only wait for a message a at its initial state. This is reflected from an internal choice $\{?a\}$ at state 2 of each minimal public view from Figure 7.11. Therefore, it is not much optimized result one can achieve by synthesizing a smaller substitute for service U_1.

In contrast to the equivalence class of service S_1, two equivalent public views of S_1 can be relatively, small as they do not impose any more restrictions on its controllers by reordering the communication events. ◁

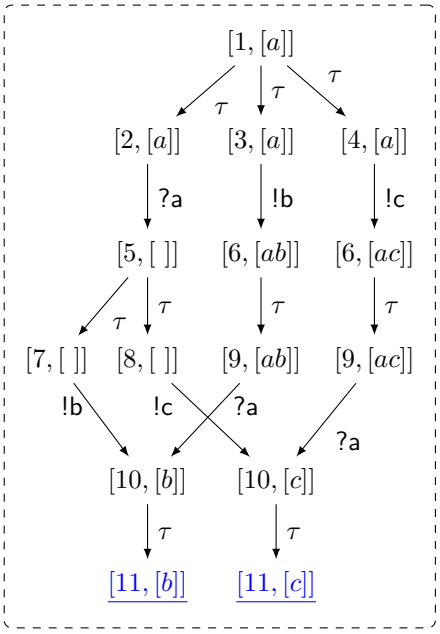
Note that the two behavioral compatibility criteria $\mathcal{B} \in \{df_k, rp_k\}$ share the same procedure for constructing a public view for a service, though the underlying artifacts



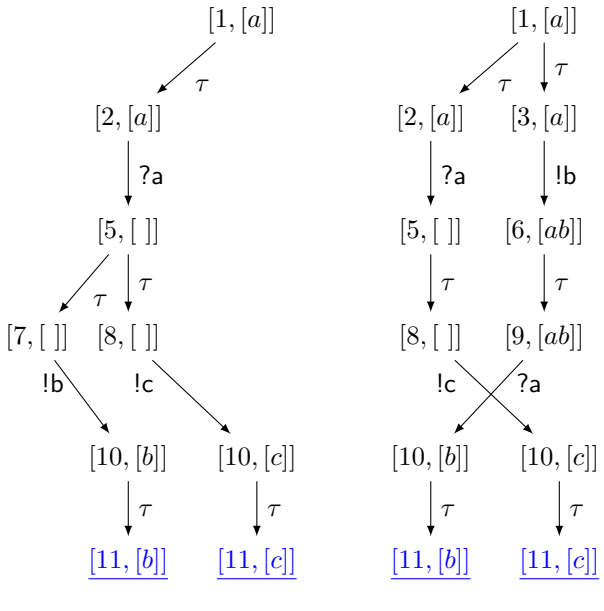
$$\widehat{\psi^*}(q_1) = \{\ \{!a\}\ \}$$
$$\widehat{\psi^*}(q_2) = \{\ \{?b\}, \{?c\}\ \}$$
$$\widehat{\psi^*}(q_3) = \{\ \{!a\}\ \}$$
$$\widehat{\psi^*}(q_4) = \{\ \{\mathit{final}\}\ \}$$
$$\widehat{\psi^*}(q_5) = \{\ \{\ \}, \{!a\}, \{?b\}, \{?c\}, \{\mathit{final}\}\ \}$$

$$Cov_{\psi^*}(\widehat{\mathcal{C}^2_{\mathcal{B}}}(U_1)) = \{ \{[2,[\,]]\}, \{[11,[b]],[11,[c]]\}, \{[9,[\,]]\}, \{[11,[\,]]\} \ \}$$

(a) $mp_{rp,k}(\widehat{\mathcal{C}^2_{df,k}}(U_1))$



(b) a transition system of $\widehat{\mathcal{C}^2_{df,k}}(U_1)$ that is associated with state q_2



(c) two non-deterministic results of marking the two situations in q_2

Figure 7.10.: A most permissive k-responsive controller $mp_{rp,k}(\widehat{\mathcal{C}^2_{rp,k}}(U_1))$ of a compact canonical k-responsive substitute $\widehat{\mathcal{C}^2_{rp,k}}(U_1)$ (from Figure 7.9) and a transition system of $\widehat{\mathcal{C}^2_{rp,k}}(U_1)$ that is associated with state q_2.

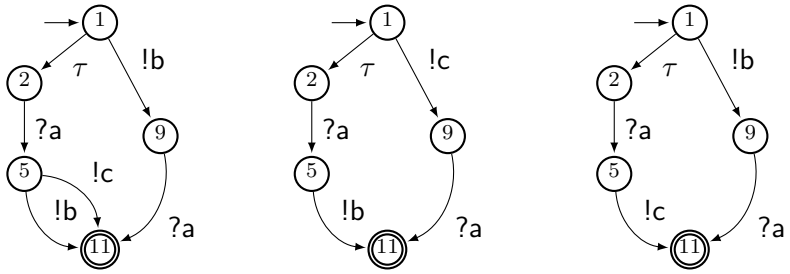


Figure 7.11.: Three minimal public views for service U_1 from Figure 7.9, each is generated by applying Algorithm 1 for message bound $k = 1$ on each message channel.

for the computation are significantly different. The Algorithm 1 is not yet implemented during the time of writing this thesis.

7.3.4. Synthesizing a Service that Preserves Selected Partners

One important application of service substitution is service improvement. Suppose a service designer wants to revise a service that, for example, is unprofitable or introduces a bottleneck into a collaboration. A service designer needs an upgraded service that preserves only its major partners, as either an enterprise no longer provides a support for other partners or other partners must adjust their services accordingly. Under some situations, a service designer also wants to support an additional service that is not previously a partner of a service. In such cases, constructing a service T that is a substitute for a given service S is possibly too restrictive, as service T must preserve *every* partner of service S according to a given criterion.

In this section, we illustrate how to apply the results from Part II to tackle the problems introduced by the following scenario. Given a $\mathcal{B}$-controllable service S and a (fixed) set $\mathcal{P}$ of $\mathcal{B}$-controllable services that are all interface equivalent to each other and each of them is interface compatible to S. Possibly, there is a service $P_i \in \mathcal{P}$ where P_i is not a $\mathcal{B}$ controller of service S for $\mathcal{B} \in \{df_k, rp_k\}$ and message bound $k \in \mathbb{N}$. This scenario is illustrated by Figure 7.12.

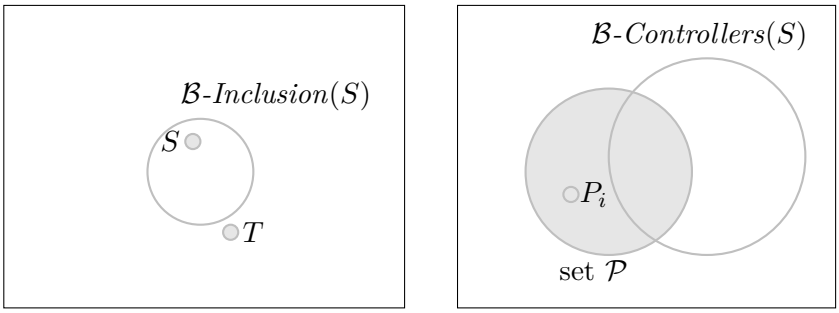


Figure 7.12.: Two services S and T with a set $\mathcal{P}$ of services where every $P_i \in \mathcal{P}$ is a $\mathcal{B}$-controller of service T but not necessarily of service S for $\mathcal{B} \in \{df_k, rp_k\}$.

The goal is to construct a service T that preserves every service in the set $\mathcal{P}$ as its $\mathcal{B}$ controller, this means, every $P_i \in \mathcal{P}$ is a $\mathcal{B}$ controller of T for $\mathcal{B} \in \{df_k, rp_k\}$. In case of deadlock freedom (i. e., $\mathcal{B} = df_k$), a more restricted version of this scenario has been proposed in Stahl et al. [2009] as *substitution under restriction* and in Stahl [2009] as *substitution under preservation*. Thereby, the set $\mathcal{P}$ must be a subset of all deadlock-free controllers of service S and the decision procedure is realized by an intersection operator on operating guideline for services [Bretschneider, 2007, Stahl et al., 2009, Kaschner and Wolf, 2009]. As the sets of deadlock-free controllers can be represented by a deadlock-free operating guideline [Lohmann et al., 2007a], Stahl et al. [2009] has suggested to use the product of deadlock-free operating guidelines of each service in the set $\mathcal{P}$. Such an approach requires to first compute an operating guideline for each service in the set $\mathcal{P}$, and then to compute their product, which represents the intersection of all deadlock-free controllers of every service in the set $\mathcal{P}$.

In this section, we first relax one requirement from Stahl et al. [2009], Stahl [2009] which states that $\mathcal{P} \subseteq \mathcal{B}\text{-}Controllers(S)$. As a result, we do not assume the set $\mathcal{P}$ to be included in $\mathcal{B}\text{-}Controllers(S)$. However, we assume that every service P_i in the set $\mathcal{P}$ must be $\mathcal{B}$-controllable, meaning it is possible to find a $\mathcal{B}$-controller of P_i. Then, we generalize the solutions for both deadlock freedom ($\mathcal{B} = df_k$) and responsiveness ($\mathcal{B} = rp_k$) criteria of service composition. To find a solution for this scenario, we target a service T that is a $\mathcal{B}$ controller of every service P_i in the set $\mathcal{P}$. Therefore, we characterize the intersection of all $\mathcal{B}$ controllers of every service P_i in the set $\mathcal{P}$, and then synthesize a targeted service T from this characterization. The set $\bigcap_{P_i \in \mathcal{P}} \mathcal{B}\text{-}Controllers(P_i)$ and a targeted service T are illustrated in Figure 7.13.

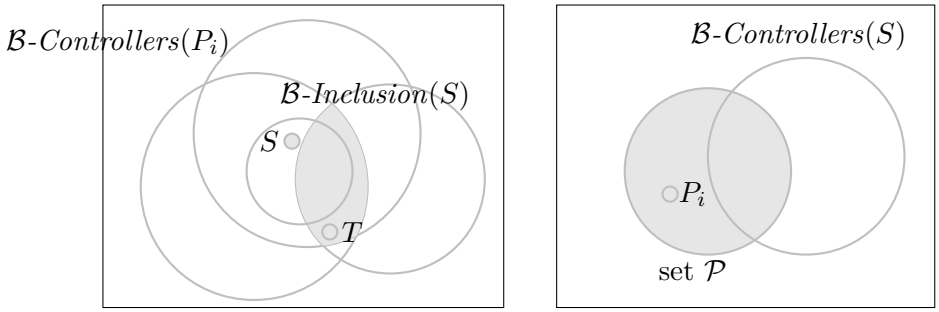


Figure 7.13.: The intersection $\bigcap_{P_i \in \mathcal{P}} \mathcal{B}\text{-}Controllers(P_i)$ of all $\mathcal{B}$-controllers of each service $P_i \in \mathcal{P}$ for $\mathcal{B} \in \{df_k, rp_k\}$.

We propose two different solutions for the sketched scenario. Each solution promises an operating guideline that characterizes the set $\bigcap_{P_i \in \mathcal{P}} \mathcal{B}\text{-}Controllers(P_i)$. As a result, it can be used to synthesize a targeted service T. Though, we will illustrate that the second solution has several advantages over the first solution.

Solution 1: The first solution is inherited from Stahl et al. [2009] and Stahl [2009]. This solution applies the product operation on the operating guidelines (cf. Section 3.2.1)to characterize the set $\bigcap_{P_i \in \mathcal{P}} \mathcal{B}\text{-}Controllers(P_i)$ in Figure 7.17. For each service

$P_i \in \mathcal{P}$, we first compute an operating guideline of P_i (cf. Section 3.3.2 in case $\mathcal{B} = df_k$ and Section 3.3.3 in case $\mathcal{B} = rp_k$). Then, we compute the product of all operating guidelines of all $P_i \in \mathcal{P}$ as a finite representation of the set $\bigcap_{P_i \in \mathcal{P}} \mathcal{B}\text{-}Controllers(P_i)$ (by applying Corollary 3.20 in case $\mathcal{B} = df_k$ and Corollary 3.21 in case $\mathcal{B} = rp_k$).

We prove the correctness of this solution in the following lemmas.

Lemma 7.4 (Intersection of deadlock-free operating guidelines of P_i).
For each message bound $k \in \mathbb{N}$ and a non-empty set $\mathcal{P}$ of k-deadlock-freely controllable services:

$$\bigcap_{P_i \in \mathcal{P}} df_k Controllers(P_i) \quad = \quad Comply_\beta(\bigotimes_{P_i \in \mathcal{P}} OG_k(P_i))$$

where $OG_k(P_i)$ is a k-deadlock-free operating guideline of P_i.

Proof. Follows from Lemma 3.51 and Corollary 3.20. □

Lemma 7.5 (Intersection of responsive operating guidelines of P_i).
For each message bound $k \in \mathbb{N}$ and a non-empty set $\mathcal{P}$ of k-responsively controllable services:

$$\bigcap_{P_i \in \mathcal{P}} rp_k Controllers(P_i) \quad = \quad Comply_\gamma(\bigotimes_{P_i \in \mathcal{P}} OG(P_i))$$

where $OG_k(P_i)$ is a k-responsive operating guideline of P_i.

Proof. Follows from Lemma 3.69 and Corollary 3.21. □

Clearly, the empty product operating guideline calculated from one of the lemma above means that the intersection set of all $\mathcal{B}$-controllers of all $P_i \in \mathcal{P}$ is empty. In such a situation, we conclude that it is not possible to construct a service T that preserves every service in the set $\mathcal{P}$ as its $\mathcal{B}$ controllers for $\mathcal{B} \in \{df_k, rp_k\}$.

In case the calculated product operating guideline is not empty, it represents the set $\bigcap_{P_i \in \mathcal{P}} \mathcal{B}\text{-}Controllers(P_i)$ for $\mathcal{B} \in \{df_k, rp_k\}$ that is a solution for the sketched scenario. Nevertheless, the product operating guideline is not a service, but a finite representation of a set of services with distinguished properties. Thus, we can apply each of two following techniques to synthesize a targeted service T from the product operating guideline. Given an operating guideline, we either

1. obtain T by taking an underlying service automaton of the operating guideline, as the underlying service automaton is described by the operating guideline (cf. Proposition 3.43 for $\mathcal{B} = df_k$ and Lemma 3.65 for $\mathcal{B} = rp_k$), or

2. obtain T by constructing either a complete canonical representative of the set (cf. Definition 3.23) or a compact canonical representative of the set (cf. Definition 3.30) from the operating guideline.

The first technique synthesizes a service T that is relatively compact in size, that is, has similar size as the operating guideline. Though, it is a member of the set $\bigcap_{P_i \in \mathcal{P}} \mathcal{B}\text{-}Controllers(P_i)$, it is not a canonical representative of the set. This means, further synthesis tasks on T may not guarantee that the result is still a valid solution for this scenario. The second technique synthesizes a service T that is a member of the set that is described by the given operating guideline (cf. Lemma 3.25 for $\mathcal{B} = df_k$ and Lemma 3.26 for $\mathcal{B} = rp_k$). Though relatively larger in size, it is a service that represents the set $\bigcap_{P_i \in \mathcal{P}} \mathcal{B}\text{-}Controllers(P_i)$ (cf. Section 4.1 and Section 5.1). In this sense, we can consider the synthesized service T as a specification for all services in the intersection set and then refine service T further e. g., by means of transformation (cf. Section 7.3.1) or by means of construction (cf. Section 7.3.2).

Consequently, T is a solution for a service that is a $\mathcal{B}$ controller of every single service P_i in the set $\mathcal{P}$ for $\mathcal{B} \in \{df_k, rp_k\}$.

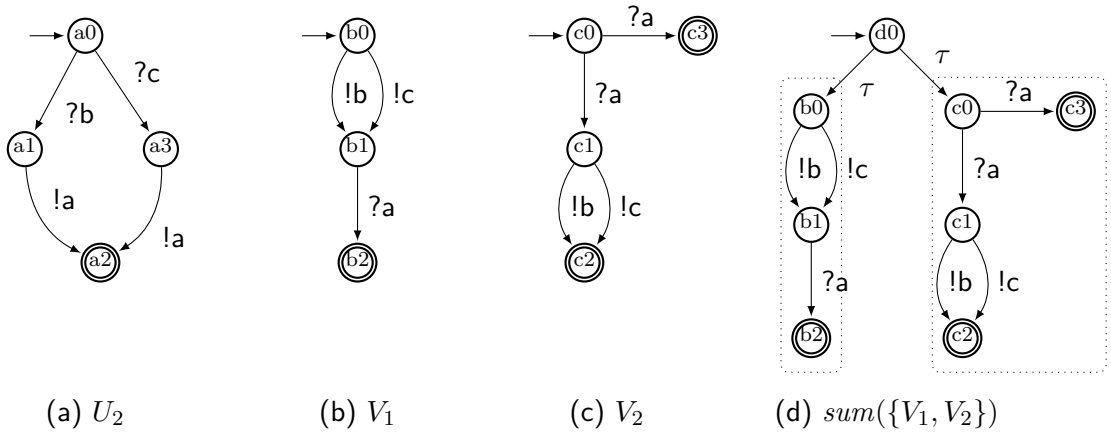
Example 7.6. Figure 7.14 illustrates four services U_2, V_1, V_2, and $sum(\{V_1, V_2\})$. Service U_2 has the interface $I = \{b, c\}$ and $O = \{a\}$. The three services V_1, V_2, and $sum(\{V_1, V_2\})$ have the same interface $I = \{a\}$ and $O = \{b, c\}$, each of which is compatible to U_2.

Consider message bound $k = 1$ on each message channel and k-responsiveness of service composition ($\mathcal{B} = rp_k$). Given a service U_2 and the two services $V_1, V_2 \in \mathcal{V}$. Service V_1 is a k-responsive controller of U_2, but obviously service V_2 is not. This is because services U_2 and V_2 both wait at their initial state to receive a message. Therefore, U_2 and V_2 cannot interact responsively with each other.

As sketched by Solution 1, we first synthesize a k-responsive operating guideline for V_1 and for V_2. The two operating guidelines $OG(V_1)$ of V_1 and $OG(V_2)$ of V_2 are illustrated in Figure 3.5 (from Section 3.2.1, Chapter 3) as $OG_1 = OG(V_1)$ and $OG_2 = OG(V_2)$. Then, we synthesize the product $OG(V_1) \otimes OG(V_2)$ of the two operating guidelines. The calculated product is illustrated in Figure 7.15 as $[W, \psi^*] = OG(V_1) \otimes OG(V_2)$. The k-responsive operating guidelines $[W, \psi^*]$ represents the set $(rp_k Controllers(V_1) \cap rp_k Controllers(V_2))$. To synthesize a service that can interact with both services V_1 and V_2, we either take the underlying service W of $[W, \psi^*]$ or construct a canonical representative from $[W, \psi^*]$. Figure 7.15 illustrates a compact canonical representative of the set $(rp_k Controllers(V_1) \cap rp_k Controllers(V_2))$ constructed from $[W, \psi^*]$. ◁

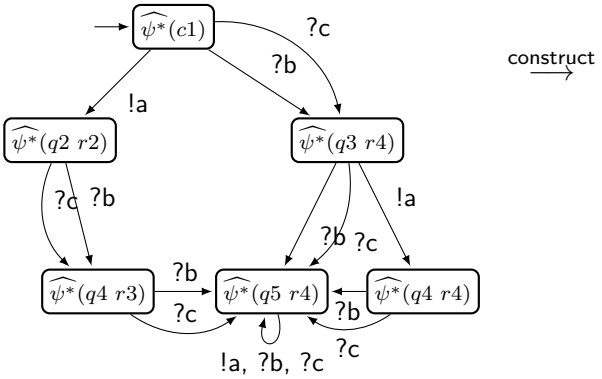
Solution 1 offers a technique to solve the sketch scenario using the product operating guideline. Though, Solution 1 can be expensive in practice. This is because we need to calculate an operating guideline for every service P_i in the given set $\mathcal{P}$ before performing the product of all calculated operating guidelines. Next, we will present an alternative solution for the sketched scenario.

Solution 2: The second solution applies the *sum* operation defined by Definition 4.78 on the set $\mathcal{S}$ of services. *sum* is an operation on a given set of services that synthesizes a service $sum(\mathcal{P})$ that offers a non-deterministic τ option to each service in the given set $\mathcal{P}$. For each service $P_i \in \mathcal{P}$, we first compute the *sum* operator to combine all services P_i in the set $\mathcal{P}$ using nondeterministic internal τ choice. Then, we compute an operating



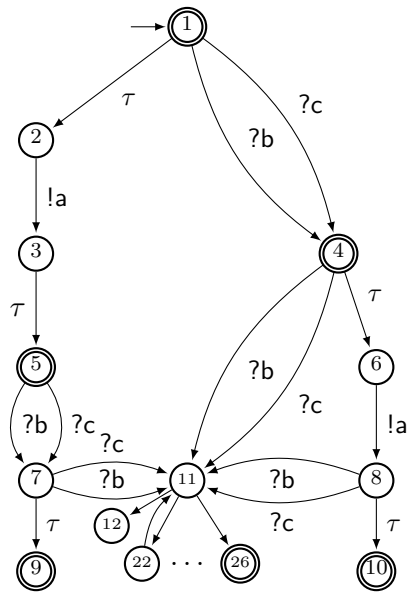
(a) U_2 (b) V_1 (c) V_2 (d) $sum(\{V_1, V_2\})$

Figure 7.14.: Four services U_2, V_1, V_2, and $sum(\{V_1, V_2\})$. Service U_2 has the interface $I = \{b, c\}$ and $O = \{a\}$. The three services V_1, V_2, and $sum(\{V_1, V_2\})$ have the same interface $I = \{a\}$ and $O = \{b, c\}$.



$$
\begin{aligned}
\widehat{\psi^*}(q1\ r1) &= \{\ \{!a\}\}, \\
\widehat{\psi^*}(q2\ r2) &= \{\ \{?b, ?c, \mathit{final}\}\ \} \\
\widehat{\psi^*}(q3\ r4) &= \{\ \{!a\}\ \} \\
\widehat{\psi^*}(q4\ r3) &= \{\ \{\mathit{final}\}\ \} \\
&= \widehat{\psi^*}(q4\ r4) \\
\widehat{\psi^*}(q5\ r4) &= \{\ \{\ \}, \{!a\}, \{?b\}, \{?c\}, \{\mathit{final}\}\ \}
\end{aligned}
$$

construct



(a) $[W, \psi^*]$ (b) W^*

Figure 7.15.: The responsive operating guideline $[W, \psi^*]$ and a compact canonical representative W^* of all service automata represented by $[W, \psi^*]$.

guideline of $sum(\mathcal{P})$ as a finite representation of the set $\bigcap_{P_i \in \mathcal{P}} \mathcal{B}\text{-}Controllers(P_i)$ for $\mathcal{B} \in \{df_k, rp_k\}$.

We prove the correctness of this solution in the following theorem.

Theorem 7.7 (Controllers of the sum of P_i).
For each message bound $k \in \mathbb{N}$ and each nonempty set $\mathcal{P}$ of services:

$$\bigcap_{P_i \in \mathcal{P}} \mathcal{B}\text{-}Controllers(P_i) \quad = \quad \mathcal{B}\text{-}Controllers(sum(\mathcal{P}))$$

for $\mathcal{B} \in \{df_k, rp_k\}$.

Proof. We prove this theorem for each $\mathcal{B} \in \{df_k, rp_k\}$.

$\mathcal{B} = df_k$: For each service $P_i \in \mathcal{P}$ holds: $sum(\mathcal{P}) \sqsubseteq_{SF} P_i$ by applying transformation rule $F(6)$ (Lemma 4.81). As stable failures refinement implies deadlock-free inclusion (Lemma 4.34), $P_i \sqsubseteq_{df,k} sum(\mathcal{P})$ follows and $df_k Controllers(P_i) \supseteq df_k Controllers(sum(\mathcal{P}))$ holds by definition.

$\mathcal{B} = rp_k$: For each service $P_i \in \mathcal{P}$ holds: $sum(\mathcal{P}) \sqsubseteq_{RF} P_i$ by applying transformation rule $F(6)$ (Lemma 4.81). As responsive failures refinement implies responsive inclusion (Lemma 4.56), $P_i \sqsubseteq_{rp,k} sum(\mathcal{P})$ follows and $rp_k Controllers(P_i) \supseteq rp_k Controllers(sum(\mathcal{P}))$ holds by definition.

With the set theory on intersection, we conclude that the theorem holds. □

Theorem 7.7 allows us to provide an alternative solution to the sketched scenario by constructing an operating guideline of the *sum* of all service P_i in the set $\mathcal{P}$. The $sum(\mathcal{P})$ is a service that offers nondeterministic internal τ options between all services in $sum(\mathcal{P})$. Clearly, if $sum(\mathcal{P})$ is not $\mathcal{B}$-controllable, then it is not possible to construct an operating guideline for $sum(\mathcal{P})$ and the set $\bigcap_{P_i \in \mathcal{P}} \mathcal{B}\text{-}Controllers(P_i)$ is an empty set. In case the set of $\bigcap_{P_i \in \mathcal{P}} \mathcal{B}\text{-}Controllers(P_i)$ is not empty, we can apply similar techniques as described in Solution 1 for synthesizing a target service T from the constructed operating guideline.

Obviously, Solution 2 is an improved solution for the sketched scenario. This is because the *sum* operator on the set of services $\mathcal{P}$ is relatively inexpensive in comparison to constructing an operating guideline for each $P_i \in \mathcal{P}$. In addition, it requires the construction of an operating guideline only once on service $sum(\mathcal{P})$.

Example 7.8. Figure 7.14(d) illustrates service $sum(\{V_1, V_2\})$ that is constructed from services V_1 and V_2. Service $sum(\{V_1, V_2\})$ offers non-deterministic τ options between services V_1 and V_2. The k-responsive operating guideline $[W, \psi^*] = OG(sum(\{V_1, V_2\}))$ for service $sum(\{V_1, V_2\})$ is illustrated in Figure 7.15 as a finite representation of the set $(rp_k Controllers(V_1) \cap rp_k Controllers(V_2))$. To synthesize a service that can interact with both services V_1 and V_2, we either take the underlying service W of $[W, \psi^*]$ or construct a canonical representative from $[W, \psi^*]$. Figure 7.15(b) illustrates a compact canonical representative W^* of the set $(rp_k Controllers(V_1) \cap rp_k Controllers(V_2))$ constructed from $[W, \psi^*]$. ◁

The related implementation of this procedure is currently under development and will be integrated into the open source software tool *Evans* [Parnjai, 2011b].

7.3.5. Synthesizing Substitutes for Multiple Services

Suppose a service designer need to either substitute multiple versions of a service, or substitute several similar services from different vendors or service providers. These different version of services have the same set of interface, but each version is not a substitute for every other service under a given substitution criterion. In such cases, constructing a service T that is a substitute for a single service is not sufficient, as service T must be able to substitute every version of a service that is given. Synthesizing a service that fulfills this constraint is not a trivial task, as it must interact properly with every controller of every given service.

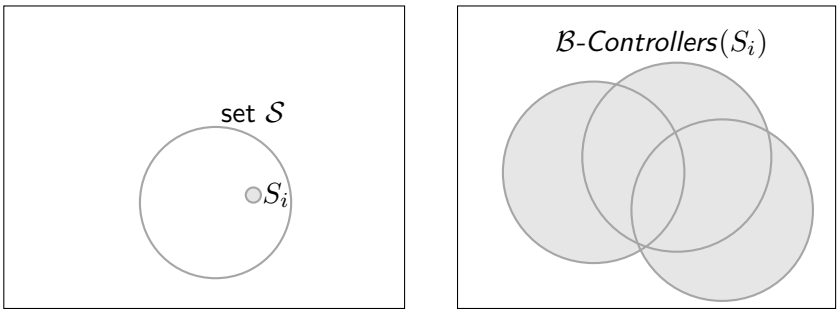


Figure 7.16.: A set $\mathcal{S}$ of services and their sets of $\mathcal{B}$ controllers for $\mathcal{B} \in \{df_k, rp_k\}$.

In this section, we illustrate how to apply the results from Part II to tackle the problems induced by the following scenario. Given a (fixed) set $\mathcal{S}$ of $\mathcal{B}$-controllable services with equivalent interface. The goal is to construct a service T that can substitute every service in the given set under $\mathcal{B}$ inclusion for $\mathcal{B} \in \{df_k, rp_k\}$. This scenario is illustrated by Figure 7.16.

To find a solution for this scenario, we target a service T which is a substitute for every service $S_i \in \mathcal{S}$. Therefore, we characterize the intersection of all substitutes for every service $S_i \in \mathcal{S}$ under $\mathcal{B}$ inclusion, and then synthesize a targeted service T from such a characterization. The set $\bigcap_{S_i \in \mathcal{S}} \mathcal{B}\text{-}Inclusion(S_i)$ and a targeted service T are illustrated in Figure 7.17.

We propose two different solutions for the sketched scenario. Each solution promises an output operating guideline that characterizes the set $\bigcap_{S_i \in \mathcal{S}} \mathcal{B}\text{-}Inclusion(S_i)$ and as a result can be use to synthesize a targeted service T. Analogously, the two solutions follow the similar idea as the solutions proposed in Section 7.3.4 for a different scenario.

Solution 1: The first solution applies the product operating guideline defined by Definition 3.19 to characterize the set $\bigcap_{S_i \in \mathcal{S}} \mathcal{B}\text{-}Inclusion(S_i)$ in Figure 7.17. For each service $S_i \in \mathcal{S}$, we first compute a canonical $\mathcal{B}$ controller $\mathcal{B}max(S_i)$ of each service $S_i \in \mathcal{S}$ (by applying either Definition 3.23 for the complete controller or Definition 3.30 for the compact controller). For each canonical controller $\mathcal{B}max(S_i)$ for $S_i \in \mathcal{S}$, we then compute an operating guideline of each service $\mathcal{B}max(S_i)$ (using Corollary 3.20 in case $\mathcal{B} = df_k$ and Corollary 3.21 in case $\mathcal{B} = rp_k$). For the final step, we compute the product of all computed operating guidelines as a finite representation of the set $\bigcap_{S_i \in \mathcal{S}} \mathcal{B}\text{-}Inclusion(S_i)$.

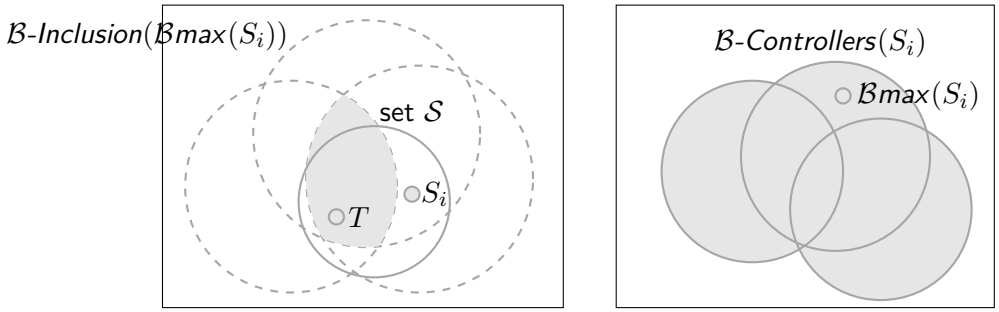


Figure 7.17.: The intersection $\bigcap_{S_i \in \mathcal{S}} \mathcal{B}\text{-}Inclusion(S_i)$ of all substitutes for every service $S_i \in \mathcal{S}$ for $\mathcal{B} \in \{df_k, rp_k\}$.

We prove the correctness of this solution in the following lemmas.

Lemma 7.9 (Intersection of deadlock-free inclusion substitutes).
For each message bound $k \in \mathbb{N}$ and a non-empty set $\mathcal{S}$ of k-deadlock-freely controllable services:

$$\bigcap_{S_i \in \mathcal{S}} df_k Inclusion(S_i) \quad = \quad Comply_\beta(\bigotimes_{S_i \in \mathcal{S}} OG(\ max_{df,k}(S_i)\ ))$$

where $OG_k(max_{df,k}(S_i))$ is a k-deadlock-free operating guideline of a canonical k-deadlock-free controller $max_{df,k}(S_i)$ of S_i.

Proof. Follows from Lemma 3.51, Corollary 3.20, and Corollary 5.23. □

Lemma 7.10 (Intersection of responsive inclusion substitutes).
For each message bound $k \in \mathbb{N}$ and a non-empty set $\mathcal{S}$ of k-responsively controllable services:

$$\bigcap_{S_i \in \mathcal{S}} rp_k Inclusion(S_i) \quad = \quad Comply_\gamma(\bigotimes_{S_i \in \mathcal{S}} OG(\ max_{rp,k}(S_i)\ ))$$

where $OG_k(max_{rp,k}(S_i))$ is a k-responsive operating guideline of a canonical k-responsive controller $max_{rp,k}(S_i)$ of S_i.

Proof. Follows from Lemma 3.69, Corollary 3.21, and Corollary 5.28. □

Clearly, an empty product operating guideline calculated from one of the lemmas above means that the intersection set of all substitutes for all $S_i \in \mathcal{S}$ is empty. Thereby, we conclude that it is not possible to construct a service T that is a substitute for every service in the set $\mathcal{S}$ under $\mathcal{B}$ inclusion for $\mathcal{B} \in \{df_k, rp_k\}$. In case the calculated product operating guideline is not empty, it represents the set $\bigcap_{S_i \in \mathcal{S}} \mathcal{B}\text{-}Inclusion(S_i)$ that is a solution for the sketched scenario. To synthesize a service T from the product operating guideline, we refer to the two techniques described in Section 7.3.4.

In practice, Solution 1 can be quite expensive, as it involves several computational steps. First, we calculate a canonical $\mathcal{B}$ controller $\mathcal{B}max(S_i)$ of each $S_i \in \mathcal{S}$. Then, we compute for each $\mathcal{B}max(S_i)$ its operating guideline before performing the product of all calculated operating guidelines. In the followings, we will describe an alternative solution that is less expensive than Solution 1, as it requires only one step of calculating an operating guideline.

Solution 2: The second solution applies the *sum* operation defined by Definition 4.78 on the set $\mathcal{S}$ of services. *sum* is an operation on a given set of service that synthesize a service that offers a non-deterministic τ option to each service in the given set. For each service $S_i \in \mathcal{S}$, we first compute a canonical $\mathcal{B}$-controller $\mathcal{B}max(S_i)$ of S_i as a canonical representative of the set of all $\mathcal{B}$-controllers of S_i (cf. Section 5.1). Then we compute the *sum* operator to combine all canonical $\mathcal{B}$-controllers $\mathcal{B}max(S_i)$ using a nondeterministic internal τ choice. Finally, we compute an operating guideline of the *sum* of all canonical controllers as a finite representation of the set $\bigcap_{S_i \in \mathcal{S}} \mathcal{B}\text{-}Inclusion(S_i)$ for $\mathcal{B} \in \{df_k, rp_k\}$.

Consider the case of deadlock freedom (i. e., $\mathcal{B} = df_k$). The set of all k deadlock-free controllers of a canonical k-deadlock-free controller of a given service S coincides with the set of all substitutes for S under k-deadlock freedom inclusion (cf. Corollary 5.23). Therefore, we can lift up the result described by Theorem 7.7 in case $\mathcal{B} = df_k$ from the set $\mathcal{P}$ to the set $Max(\mathcal{S}) = \{max_{df,k}(S) \mid S_i \in \mathcal{S}\}$. This means, we characterize the set $\bigcap_{S_i \in \mathcal{S}} df_k Inclusion(S_i)$ in terms of the set $df_k Controllers(\ sum(Max(\mathcal{S}))\ )$ with the following theorem.

Theorem 7.11 (Characterizing intersection with sum of canonical deadlock-free controllers).
For each message bound $k \in \mathbb{N}$ and each controllable nonempty set $\mathcal{S}$ of services:

$$\bigcap_{S_i \in \mathcal{S}} df_k Inclusion(S_i) \quad = \quad df_k Controllers(\ sum(Max(\mathcal{S}))\ )$$

where $Max(\mathcal{S}) = \{max_{df,k}(S_i) \mid S_i \in \mathcal{S}\}$.

Proof. Follows from Corollary 5.23 and Theorem 7.7 in case $\mathcal{B} = df_k$. □

To realize the result from Theorem 7.11, we apply the *sum* operator defined by Definition 4.78 to combine all services in the set $Max(\mathcal{S})$ using a nondeterministic internal τ option. Using Theorem 7.7 for $\mathcal{B} = df_k$, we can compute a deadlock-free operating guideline of $sum(Max(\mathcal{S}))$ as a finite representation of the set $\bigcap_{S_i \in \mathcal{S}} df_k Inclusion(S_i)$. Clearly, an empty deadlock-free operating guideline of $sum(Max_k(\mathcal{S}))$ means that the set $\bigcap_{S_i \in \mathcal{S}} df_k Inclusion(S_i)$ is empty and it is not possible to construct a service T that is a substitute for every single service in the set $\mathcal{S}$ under k-deadlock freedom inclusion.

Consider the case of responsiveness (i. e., $\mathcal{B} = rp_k$). The set of all k-responsive controllers of a canonical k-responsive controller of a given service S coincides with the set of all substitutes for S under k-responsiveness inclusion (cf. Corollary 5.23). Similar to

the case of deadlock freedom, we can lift up the result described by Theorem 7.7 in case $\mathcal{B} = rp_k$ from the set $\mathcal{P}$ to the set $Max(\mathcal{S}_\rangle) = \{max_{rp,k}(S) \mid S_i \in \mathcal{S}\}$. This means, we characterize the set $\bigcap_{S_i \in \mathcal{S}} rp_k Inclusion(S_i)$ in terms of the set $rp_k Controllers(\ sum(Max(\mathcal{S}))\ )$ with the following theorem.

Theorem 7.12 (Characterizing intersection with sum of canonical responsive controllers).
For each message bound $k \in \mathbb{N}$ and each controllable nonempty set $\mathcal{S}$ of services:

$$\bigcap_{S_i \in \mathcal{S}} rp_k Inclusion(S_i) \quad = \quad rp_k Controllers(\ sum(Max(\mathcal{S}))\ )$$

where $Max(\mathcal{S}) = \{max_{rp,k}(S_i) \mid S_i \in \mathcal{S}\}$.

Proof. Follows from Corollary 5.28 and Theorem 7.7 in case $\mathcal{B} = rp_k$. □

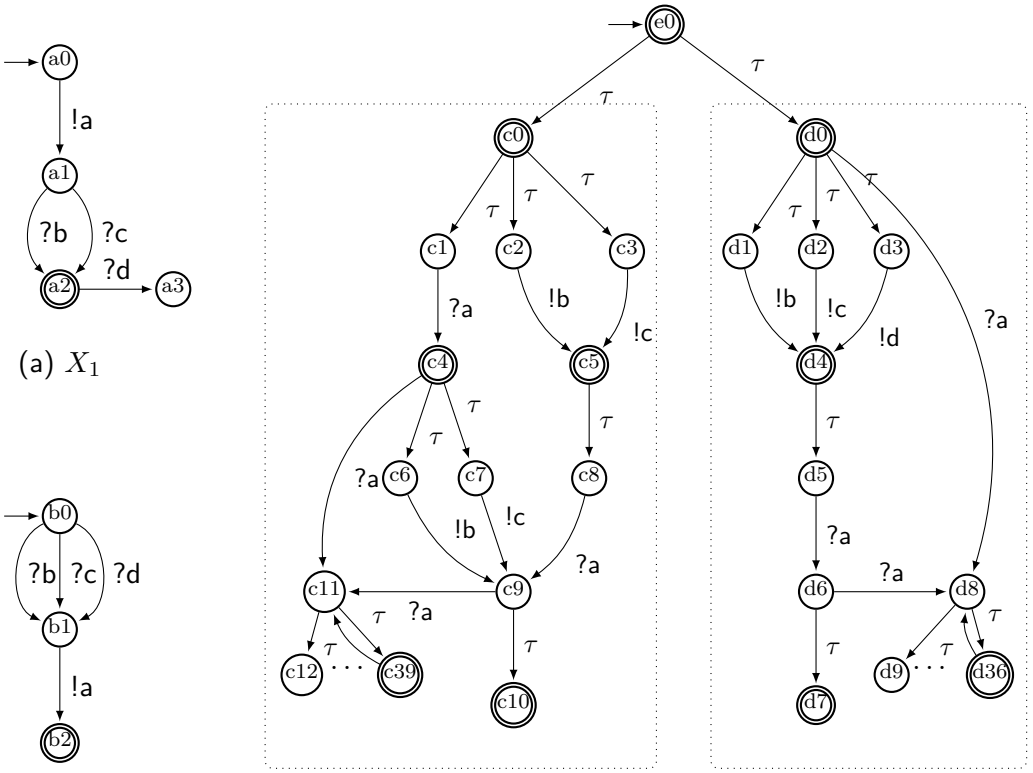
To realize the result from Theorem 7.12, we apply the *sum* operator defined by Definition 4.78 to combine all services in the set $Max(\mathcal{S})$ using a nondeterministic internal τ option. Using Theorem 7.7 for $\mathcal{B} = rp_k$, we can compute a responsive operating guideline of $sum(Max(\mathcal{S}))$ as a finite representation of the set $\bigcap_{S_i \in \mathcal{S}} rp_k Inclusion(S_i)$. Clearly, an empty responsive operating guideline of $sum(Max(\mathcal{S}))$ means that the set $\bigcap_{S_i \in \mathcal{S}} rp_k Inclusion(S_i)$ is empty and it is not possible to construct a service T that is a substitute for every single service in the set $\mathcal{S}$ under k-responsiveness inclusion.

For both solutions, the computed operating guideline of $sum(Max(\mathcal{S}))$ is not a service, but a finite representation of the set $\bigcap_{S_i \in \mathcal{S}} \mathcal{B}\text{-}Inclusion(S_i)$ of services for each compatibility criterion $\mathcal{B} \in \{df_k, rp_k\}$. Nevertheless, we can synthesize a service T from the non-empty operating guideline of $sum(Max(\mathcal{S}))$ by applying similar techniques as described for Scenario 1. Consequently, T is a solution for deriving a service that can substitute every single service S_i in the set $\mathcal{S}$ under a given behavioral compatibility criterion $\mathcal{B} \in \{df_k, rp_k\}$.

The related implementation of this procedure is currently under development and will be integrated into in the the open source software tool *Evans* [Parnjai, 2011b].

Example 7.13. Figure 7.18 illustrates three services; two services X_1, X_2, and service $sum(\ \{\widehat{\mathcal{C}}_{rp,k}(X_1), \widehat{\mathcal{C}}_{rp,k}(X_2)\}\ )$ that is synthesized from X_1 and X_2. The two services X_1 and X_2 have the same interface $I = \{b, c, d\}$ and $O = \{a\}$. The service $sum(\ \{\widehat{\mathcal{C}}_{rp,k}(X_1), \widehat{\mathcal{C}}_{rp,k}(X_2)\}\ )$ is compatible to service X_1 and X_2, and it has the interface $I = \{a\}$ and $O = \{b, c, d\}$.

Consider message bound $k = 1$ on each message channel and k-responsiveness of service composition ($\mathcal{B} = rp_k$). Service X_1 is not a substitute for X_2 under k-responsive inclusion. This is because there is a controller of X_2 which sends message d before receiving message a, whereas X_1 is not able to receive message d properly after sending out message a. Service X_2 is also not a substitute for X_1 under k-responsive inclusion. This is because there is a controller of X_1 which waits for message a before sending



(a) X_1

(b) X_2

(d) $sum(\ \{\widehat{\mathcal{C}}_{rp,k}(X_1), \widehat{\mathcal{C}}_{rp,k}(X_2)\}\ )$

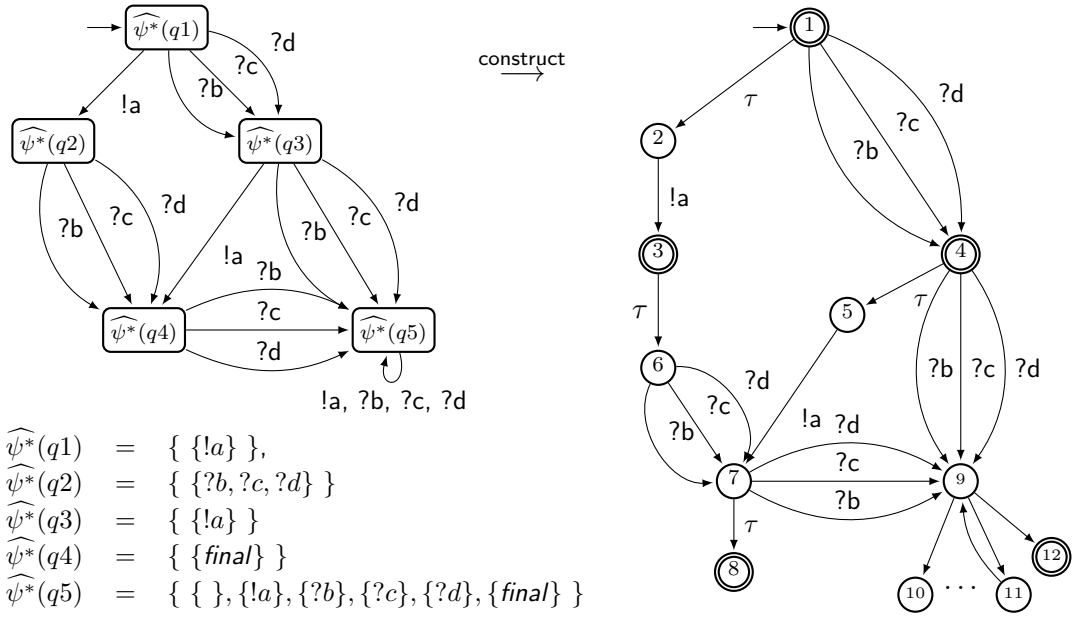
Figure 7.18.: Three services X_1, X_2, and $sum(\ \{\widehat{\mathcal{C}}_{rp,k}(X_1), \widehat{\mathcal{C}}_{rp,k}(X_2)\}\ )$. The two services X_1 and X_2 have the same interface $I = \{b, c, d\}$ and $O = \{a\}$. Service $sum(\ \{\widehat{\mathcal{C}}_{rp,k}(X_1), \widehat{\mathcal{C}}_{rp,k}(X_2)\}\ )$ has the interface $I = \{a\}$ and $O = \{b, c, d\}$.

out any message, whereas X_2 also waits for either message b, c, or d before sending out message a. Obviously, neither one of the two services X_1 and X_2 is a substitute for the other under k-responsiveness inclusion.

To synthesize a service that is a substitute for both X_1 and X_2 under k-responsiveness inclusion, service $sum(\ \{\widehat{\mathcal{C}}_{rp,k}(X_1), \widehat{\mathcal{C}}_{rp,k}(X_2)\}\ )$ is first synthesized from services $\widehat{\mathcal{C}}_{rp,k}(X_1)$ (depicted within the left dotted rectangle) and $\widehat{\mathcal{C}}_{rp,k}(X_2)$ (depicted within the right dotted rectangle) by applying the *sum* operator on the set $\{\widehat{\mathcal{C}}_{rp,k}(X_1), \widehat{\mathcal{C}}_{rp,k}(X_2)\}$.

By construction, service $sum(\ \{\widehat{\mathcal{C}}_{rp,k}(X_1), \widehat{\mathcal{C}}_{rp,k}(X_2)\}\ )$ offers a non-deterministic τ option between services $\widehat{\mathcal{C}}_{rp,k}(X_1)$ and $\widehat{\mathcal{C}}_{rp,k}(X_2)$ which are compact k-responsive controllers of X_1 and X_2 respectively. As a compact k-responsive controller of a service is a solution for its maximal k-responsive controller, $\widehat{\mathcal{C}}_{rp,k}(X_1)$ and $\widehat{\mathcal{C}}_{rp,k}(X_2)$ are solutions of maximal k-responsive controllers of X_1 and of X_2 respectively.

A k-responsive operating guideline of service $sum(\ \{\widehat{\mathcal{C}}_{rp,k}(X_1), \widehat{\mathcal{C}}_{rp,k}(X_2)\}\ )$ is illustrated in Figure 7.19 as $[Y, \psi^*]$. The operating guideline $[Y, \psi^*]$ is a finite representation of the



(a) $[Y, \psi^*]$ (b) Y^*

Figure 7.19.: The responsive operating guidelines $[Y, \psi^*]$ and a compact canonical representative Y^* of all service automata represented by $[Y, \psi^*]$.

set $(rp_k Inclusion(X_1) \cap rp_k Inclusion(X_2))$. To synthesize a service that is a substitute for both services X_1 and X_2, we either take the underlying service Y of $[Y, \psi^*]$ or construct a canonical representative from $[Y, \psi^*]$. Figure 7.15 illustrates a compact canonical representative Y^* of the set $(rp_k Inclusion(X_1) \cap rp_k Inclusion(X_2))$ as constructed from $[Y, \psi^*]$. Therefore, Y^* is a substitute for both services X_1 and X_2 and can be synthesized from the given services X_1 and X_2. $\lhd$

7.4. Repairing Incorrect Services

Given two services S and T, the question of whether service T is a substitute for service S under a given criterion is already answered by applying the techniques described in Section 7.1 and Section 7.2. In case service T is not a substitute for service S under a given criterion, a decision procedure returns *no*. In many circumstances, a service designer wants to repair the incorrect service T instead of discarding T completely, as redesigning a new and correct service from scratch is probably not a feasible option. Nevertheless, the manual correction of service T is a tedious and error-prone task, as it involves the manual interference with the service composition. To repair the behavior of service T, a service designer may either insert new correct behavior into T or remove undesirable behavior from T. In a worst case scenario, every behavior of service T is undesirable and as a result must be removed from T.

In this section, we presents techniques for correcting service behaviors in the following three scenarios for a behavioral compatibility criterion $\mathcal{B} \in \{df_k, rp_k\}$.

Scenerio 1 : Given two services P and R that are interface equivalent, where service P is a $\mathcal{B}$-controller of S but service R is not. This means, service R is not a substitute for service P, as the composition of S as R does not satisfy the given behavioral compatibility $\mathcal{B}$ of service composition.

We target a service R' that is an improved version of service R and that is a $\mathcal{B}$-controller of service S.

Scenerio 2 : Given two services S and T with the same interface where service T is not a substitute for service S under $\mathcal{B}$ inclusion. This means, there is at least one $\mathcal{B}$ controller of S which is not a $\mathcal{B}$-controller of T.

We target a service T' that is an improved version of service T and that is a substitute for service S under $\mathcal{B}$ inclusion.

Scenerio 3 : Given two services S and T with the same interface, but service T is not a substitute for service S under $\mathcal{B}$ equivalence. This means, there is at least one $\mathcal{B}$ controller of S which is not a $\mathcal{B}$-controller of T and there is at least one $\mathcal{B}$ controller of T that is not a $\mathcal{B}$-controller of S.

We target a service T' that is an improved version of service T and that is a substitute for service S under $\mathcal{B}$ equivalence.

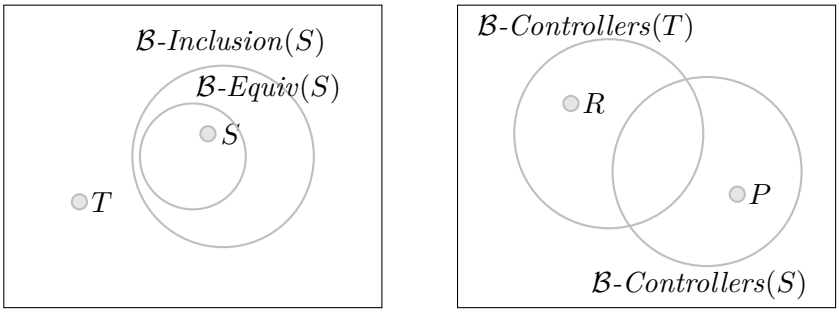


Figure 7.20.: Four services S, T, P, and R in the three scenarios.

In the remainder of this section, we present techniques for dealing with the problems described in the three scenarios above. To tackle problems described by Scenario 1 and Scenario 2, Section 7.4.1 describes a procedure based on the simulation-based graph edit distance inherited from the literature [Lohmann, 2008a] and Section 7.4.2 describes the filtering technique to remove incorrect communication behavior from an incorrect service. Finally, Section 7.4.3 presents a procedure to synthesize a targeted service described by Scenario 3.

7.4.1. Using Simulation-based Graph Edit Distance

In this section, we combine existing work on operating guidelines techniques for the correction of service behavior with our results from Part II to provide solutions for the problems described by Scenario 1 and Scenario 2.

To correct a detected error of a service in a service choreography, Lohmann [2008a] has proposed the simulation-based graph edit distance approach to compute edit actions that are necessary to repair an incorrect service to interact with others in a compatible manner. Given a behavioral compatibility criterion $\mathcal{B}$ of service composition and two interface compatible services S and P where P is a not $\mathcal{B}$ controller of S, this technique computes service P' such that P' is a $\mathcal{B}$ controller of S described by an operating guideline of S, and P' is the most similar to S based on a quantitative similarity measure defined by a simulation relation between services. This technique then describes the edit actions that are necessary to transform P into P', such as inserting new states, removing transitions, and changing transition labels. To this end, service P' is a $\mathcal{B}$ controller of service S that is most similar to service P according to the similarity measures, and P' is indeed a solution for the problem described by Scenario 1.

With the simulation-based graph edit distance approach, we are able to combine our results on characterizing the set of all substitutes for a given service S under $\mathcal{B}$ inclusion to address the problem described by Scenario 2. Given a behavioral compatibility criterion $\mathcal{B}$ of service composition and service S, we compute a finite representation of the set of all $\mathcal{B}$ controllers of S as an operating guideline of a canonical $\mathcal{B}$-controller of S (cf. Section 5.2.1). Given two services S and T where T is not a substitute for S under $\mathcal{B}$ inclusion, the errors in T can be detected by checking if T is described by the computed operating guideline. To correct a detected error of service T, we employ the simulation-based graph edit distance technique to compute service T' such that T' is a substitute for S under $\mathcal{B}$ inclusion as described by the operating guideline, and T' is most similar to T based on a quantitative similarity measure. We use the computed edit actions to transform T into T'. Therefore, service T' is a substitute for service T that is a solution for the problem described by Scenario 2.

With the help from the operating guidelines approach and the simulation-based graph edit distance approach, we are able to detect and repair an incorrect service that is not a substitute for a given service by performing various edit actions that are necessary to synthesize a substitute that is correct by design. We use the tool *Evans* [Parnjai, 2011b] to synthesize the desired operating guideline of a canonical controller and the tool *Rebecca* [Mennicke and Lohmann, 2009] to provide correct information and recommended actions needed to edit an incorrect service toward compatibility of service composition. So far the simulation-based graph edit distance approach provides solutions for only Scenario 1 and Scenario 2. Further, the approach is restricted to a service that is non-deterministic and contains no cycles.

In the subsequent section, we propose an alternative technique for correcting service behavior that restricts the edit actions to *removal* action only. Nevertheless, the technique provides the support for all three scenarios for correcting services including with non-deterministic and cyclic behaviors.

7.4.2. Using Filtering Inclusion Guidelines

In this section, we present a technique to repair service behavior by removing incorrect behavior from a given service. This repair method aims to *enforce* the correct communication behavior of a given service and, by doing so, *exclude* all communication behaviors that are not correct according a correctness criterion. Though this technique is restricted to repair only service with deterministic behavior. We refer to this technique for repairing services as *filtering* [Parnjai, 2011a]. To repair an incorrect service using a filter operation, we address the two following questions:

1. Is it possible to filter a given service X by removing communication behavior and derive a non-empty and correct service X' from service X ?

2. How to filter service X by removing as few communication behavior from service X as possible ?

The first question concerns whether there exists a *filter* operation on a given service, and the second question addresses the problem of how to apply such a *filter* operation on a given service.

Though restricted to only removing incorrect behavior, the filtering technique is useful in several scenarios. First, the correction of service behavior is a tedious and error-prone task. The removal of incorrect behavior helps to promote the reusability and rapid development of services, therefore, shorten the development cycle and the delivery time to market. Furthermore, there are circumstances where the costs and revenues for running a service must be take into account. For instance, a service designer may need to decide service substitutability based on cost functions and constraints [Gierds, 2008b, Gierds and Sürmeli, 2010]. By removing incorrect activities from a service, it is likely that the transformed service will incur less cost and yield more revenue for operating a service.

Filtering Non-Controller

In this section, we present a technique called *filtering guidelines* as a solution for scenario 1. Given service R that is deterministic and not a $\mathcal{B}$-controller of S. We target a service R' as an improved version of service R that is a $\mathcal{B}$-controller of service S. A targeted service R' is a service in which the incorrect sequences of communication events and choices have been removed from R. Technically, we can describe the set of all sequences of communication events of a given service R by using its traces and represent a service with less sequences of communication events as a service that refines R under traces (cf. Section 4.2). This allows us to describe the targeted service R' as a service that refines service R under traces (i. e., R' has less or the same traces as R), yet R' is a $\mathcal{B}$-controller of S.

For this purpose, we define a $\mathcal{B}$-filtered controller of service S with respect to service R as follows.

Definition 7.14 (Filtered controller).
Let $k \in \mathbb{N}$ be a message bound on each message channel. Let R and R' be two interface equivalent service automata. Let S be a $\mathcal{B}$ controllable service automaton that has compatible interface with R and R'. Then, service R' is a *$\mathcal{B}$-filtered controller* of service S with respect to service R iff R' refines R under traces and R' is a $\mathcal{B}$-controller of S.

The set of all $\mathcal{B}$-filtered controllers of service S with respect to service R is denoted by $\mathcal{B}\text{-}f\text{-}Controllers(R, S) = tr\text{-}refine(R) \cap \mathcal{B}\text{-}Controllers(S)$ for $\mathcal{B} \in \{df_k, rp_k\}$.

Clearly, $\mathcal{B}\text{-}f\text{-}Controllers(R, S)$ is possibly an empty set, meaning there is no filter for a controller of service R with respect to service S. The set $\mathcal{B}\text{-}f\text{-}Controllers(R, S)$ and a targeted service R' are illustrated in Figure 7.21.

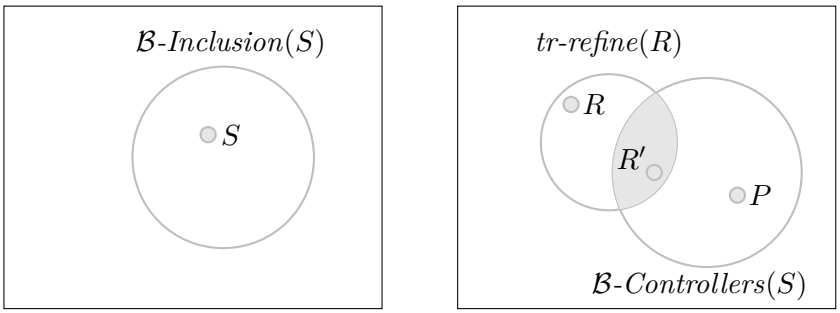


Figure 7.21.: The intersection represents the set $\mathcal{B}\text{-}f\text{-}Controllers_k(R, S)$ of all $\mathcal{B}$-filtered controllers of service R with respect to service S.

Technically, we can represent the set of all targeted services using the intersection operator of two sets of services; these are, the set of all services that refines service R under traces and the set of all $\mathcal{B}$-controllers of services S.

There are several ways to represent the set of all trace preordered set of services. The language for service automaton S is one of the natural candidates to represent the set of all services that refine S under traces. Nevertheless, we would like to employ the operating guidelines-based techniques from Lohmann et al. [2007a], Stahl et al. [2009] to characterize the intersection of two service sets using the product of annotated service automata. Therefore, the two sets must be represented as annotated service automata.

For this purpose, we require an annotated service automaton for representing the set of all traces of service. As introduced in Section 3.1, we distinguish two types of annotated service automata which represent traces of a service. Given service R, we distinguish between the *strong liberal* service $\mathcal{L}_\beta(R)$ of R as an annotated service automaton that characterizes all services that refine R under traces using the *strong compliance* relation and the *strong liberal* service $\mathcal{L}_\gamma(R)$ of R that characterizes all services that refine R under traces using the *structural compliance* relation and the *structural liberal* service.

First, we define a *strong liberal service* $\mathcal{L}_\beta(R)$ of service R as a deterministic service automaton which *strongly simulates* R.

Definition 7.15 (Construction of $\mathcal{L}_\beta(R)^\varphi$).
Let $R = [Q_R, q_{0R}, I_R, O_R, \rightarrow_R, \Omega_R]$ be a deterministic service automaton. A *strong liberal service* of R is a deterministic service automaton $\mathcal{L}_\beta(R) = [Q, q_0, I, O, \rightarrow, \Omega]$ with $I = I_R$ and $O = O_R$. Let $\varrho \subseteq Q_R \times Q$ be a binary relation between states of R and states of $\mathcal{L}_\beta(R)$. The ϱ, Q, q_0, Ω, and $\rightarrow$ are inductively defined as follows :

Base: let $[q_{0R}, q_0] \in \varrho$,

Step: if $[q_R, q] \in \varrho$ then all the followings must be satisfied

– $q \xrightarrow{\tau} q$;

– if either $q_R \in \Omega_R$ or there exists $q'_R \in \Omega_R$ with $q_R \stackrel{\epsilon}{\Rightarrow} q'_R$ and $q'_r \in \Omega_R$, then $q \in \Omega$;

– if there exists $q_R \xrightarrow{\tau}_R q'_R$, then $[q'_R, q] \in \varrho$;

– if there exists $q_R \xrightarrow{m}_R q'_R$ and $m \neq \tau$, then there exists also $q \xrightarrow{m} q'$ and $[q'_R, q'] \in \varrho$.

For each state q of a strong liberal service $\mathcal{L}_\beta(R)$ of service R, the set of all *choices* of state q is denoted by $\varphi(q) = \{ch \mid ch \subseteq \mathit{enable}(q)\}$.

The following lemma shows that a choice annotated strong liberal service $\mathcal{L}_\beta(R)^\varphi$ of service R characterizes the set *tr-refine*(R) of all traces of R using the strong compliance relation (cf. Definition 3.15).

Lemma 7.16 (Trace and annotated strong liberal service automaton).
For each service R holds: $\mathit{Comply}_\beta(\mathcal{L}_\beta(R)^\varphi) = \mathit{tr\text{-}refine}(R)$.

Proof. We will prove this lemma in two directions.

$\subseteq$: Suppose $S \in \mathit{Comply}_\beta(\mathcal{L}_\beta(R)^\varphi)$; i. e., $\mathcal{L}_\beta(R)$ strongly simulates S and for each simulated pair $[q_S, q]$ there exists $ch \in \varphi(q)$ such that state q_S strongly corresponds to ch. Consider $\sigma \in \mathit{traces}(S)$. We have $\sigma \in \mathit{traces}(\mathcal{L}_\beta(R))$ because $\mathcal{L}_\beta(R)$ simulates S. Because $\mathit{traces}(R) = \mathit{traces}(\mathcal{L}_\beta(R))$ by construction, it follows that $\sigma \in \mathit{traces}(R)$ holds and $\mathit{traces}(S) \subseteq \mathit{traces}(R)$ follows. Thus, $S \in \mathit{tr\text{-}refine}(R)$ holds.

$\supseteq$: Suppose $S \in \mathit{tr\text{-}refine}(R)$; i. e., $\mathit{traces}(S) \subseteq \mathit{traces}(R)$ holds. Because R is deterministic by assumption and $\mathcal{L}_\beta(R)$ is deterministic by construction, $\mathcal{L}_\beta(R)$ strongly simulates R and it follows that $\mathcal{L}_\beta(R)$ also strongly simulates S. Because $\varphi(q)$ at state q contains all possible choices that are a combination of outgoing events at state q including the empty choice and the choice that describes the final event *final*, there is always $ch \in \varphi(q)$ such that q_S strongly corresponds to ch. Thus, $S \in \mathit{Comply}_\beta(\mathcal{L}_\beta(R)^\varphi)$ holds. □

Next, we define a *structural liberal service* $\mathcal{L}_\gamma(R)$ of a service R as a deterministic service automaton such that $\mathcal{L}_\gamma(R)$ has the same interface as S, the same set of traces as R, and *structurally simulates* R.

Definition 7.17 (Construction of $\mathcal{L}_\gamma(R)^{\psi^*}$).
Let $R = [Q_R, q_{0R}, I_R, O_R, \rightarrow_R, \Omega_R]$ be a deterministic service automaton. A *structural liberal* service of R is a deterministic service automaton $\mathcal{L}_\gamma(R) = [Q, q_0, I, O, \rightarrow, \Omega]$ with $I = I_R$ and $O = O_R$. Let $\varrho \subseteq Q_R \times Q$ be a binary relation between states of R and states of $\mathcal{L}_\gamma(R)$. The ϱ, Q, q_0, Ω, and $\rightarrow$ are inductively defined as follows :

Base: let $[q_{0R}, q_0] \in \varrho$,

Step: if $[q_R, q] \in \varrho$ then all the followings hold:

– if either $q_R \in \Omega_R$ or there exists $q'_R \in \Omega_R$ with $q_R \stackrel{\epsilon}{\Rightarrow} q'_R$ and $q'_r \in \Omega_R$, then $q \in \Omega$;

– if there exists $q_R \stackrel{\tau}{\rightarrow}_R q'_R$ then $[q'_R, q] \in \varrho$;

– if there exists $q_R \stackrel{m}{\rightarrow}_R q'_R$ and $m \neq \tau$ then there exists also $q \stackrel{m}{\rightarrow} q'$ and $[q'_R, q'] \in \varrho$.

For each state q of a strong liberal service $\mathcal{L}_\beta(R)$ of service R, the set of all *choices* of state q is denoted by $\psi^*(q) = \{ch \mid ch \subseteq \mathit{enable}(q)\}$.

Note, that the difference between the two construction procedures of $\mathcal{L}_\beta(R)^\varphi$ and of $\mathcal{L}_\gamma(R)^{\psi^*}$ is that $\mathcal{L}_\gamma(R)^{\psi^*}$ does not have a self τ loop at every state q, unlike $\mathcal{L}_\beta(R)^\varphi$. The following lemma shows that an annotated structural liberal service $\mathcal{L}_\gamma(R)^{\psi^*}$ of service R characterizes the set *tr-refine*(R) of all traces of R using the structural compliance relation (cf. Definition 3.17).

Lemma 7.18 (Trace and annotated structural liberal service automaton).
For each service R holds: $\mathit{Comply}_\gamma(\mathcal{L}_\gamma(R)^{\psi^*}) = \mathit{tr\text{-}refine}(R)$.

Proof. We will prove this lemma in two directions.

$\subseteq$: Suppose $S \in \mathit{Comply}_\gamma(\mathcal{L}_\gamma(R)^{\psi^*})$; i. e., $\mathcal{L}_\gamma(R)$ structurally simulates S and for each simulated pair (q_S, q) there exists $ch \in \psi^*(q)$ such that q_S structurally corresponds to ch. Consider $\sigma \in \mathit{traces}(S)$. We have $\sigma \in \mathit{traces}(\mathcal{L}_\gamma(R))$ because $\mathcal{L}_\gamma(R)$ simulates S. Because $\mathit{traces}(R) = \mathit{traces}(\mathcal{L}_\beta(R))$ by construction, it follows that $\sigma \in \mathit{traces}(R)$ holds and $\mathit{traces}(S) \subseteq \mathit{traces}(R)$ follows. Thus, $T \in \mathit{tr\text{-}refine}(R)$ holds.

$\supseteq$: Suppose $S \in \mathit{tr\text{-}refine}(R)$; i. e., $\mathit{traces}(S) \subseteq \mathit{traces}(R)$ holds. Because $\mathcal{L}_\gamma(R)$ is deterministic by construction, $\mathcal{L}_\gamma(R)$ structurally simulates S and it follows that $\mathcal{L}_\gamma(R)$ also simulates S. Because $\psi^*(q)$ at a state q contains all possible choices that are a combination of outgoing events at state q including the empty choice and the choice that describes and the final event *final*, there is always $ch \in \varphi(q)$ such that q_S structurally corresponds to ch. Thus, $S \in \mathit{Comply}_\gamma(\mathcal{L}_\gamma(R)^{\psi^*})$ holds. □

Example 7.19. Figure 7.22 illustrates service U_3 and its annotated structural liberal service automaton $\mathcal{L}_\gamma(U_3)^{\psi^*}$ constructed from U_3 by applying Definition 7.17. The dashed lines represent the structural simulation between services U_3 and $\mathcal{L}_\gamma(U_3)^{\psi^*}$. The constructed $\mathcal{L}_\gamma(U_3)^{\psi^*}$ represents the set of services that refines U_3 under traces. ◁

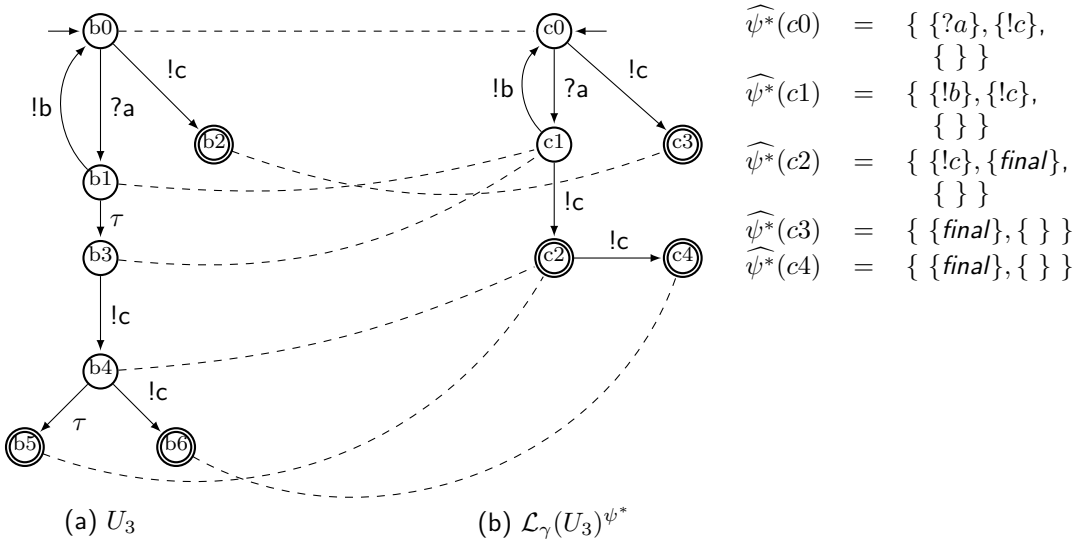


Figure 7.22.: Service U_3 and its annotated structural liberal service automaton $\mathcal{L}_\gamma(U_3)^{\psi^*}$, each with the interface $I = \{a\}$ and $O = \{b, c\}$.

In case of deadlock freedom ($\mathcal{B} = df_k$), the set of all k-deadlock-free filtered controllers of S with respect to R can be characterized by the product of two annotated service automata; firstly, an annotated strong liberal service automaton $\mathcal{L}_\beta(R)^\varphi$ of service R representing the set of all services that refine service R under traces, and secondly, the deadlock-free operating guideline $mp_{df,k}(S)^\varphi$ of service S representing the set of all k-deadlock-free controllers of service S. The correctness of this construction is asserted by the following lemma.

Lemma 7.20 (Characterization of deadlock-free filtered controllers).
For each message bound $k \in \mathbb{N}$ and each two interface compatible services R and S:

$$df_k\text{-}f\text{-}Controllers(R, S) = Comply_\beta(\mathcal{L}_\beta(R)^\varphi \otimes mp_{df,k}(S)^\varphi).$$

Proof. $df_k\text{-}f\text{-}Controllers(R, S) = tr\text{-}refine(R) \cap df_k Controllers(S)$ [by definition]
$= Comply_\beta(\mathcal{L}_\beta(R)^\varphi) \cap Comply_\beta(mp_{df,k}(S)^\varphi)$ [Lemma 7.16 and Lemma 3.51]
$= Comply_\beta(\mathcal{L}_\beta(R)^\varphi \otimes mp_{df,k}(S)^\varphi)$. [by Corollary 3.20] □

Clearly, an empty set characterized by the synthesized the filtering guideline $\mathcal{L}_\beta(R)^\varphi \otimes mp_{df,k}(S)^\varphi$ means that it is not possible to repair the behavior of service R by removing communication behavior such that the resulting service is a deadlock-free controller of S.

The filtering guideline $\mathcal{L}_\beta(R)^\varphi \otimes mp_{df,k}(S)^\varphi$ is not a service, but an annotated service automaton that represents a finite representation of the set $df_k\text{-}f\text{-}Controllers(R, S)$. In case the constructed filter is not empty, we can apply similar technique as described in Section 7.3.4 to synthesize R' from the annotated service automaton $\mathcal{L}_\beta(R)^\varphi \otimes$

$mp_{df,k}(S)^{\varphi}$. That is, we obtain R' either (1) by taking an underlying service automaton of the filter $\mathcal{L}_{\beta}(R)^{\varphi} \otimes mp_{df,k}(S)^{\varphi}$, or (2) by constructing from $\mathcal{L}_{\beta}(R)^{\varphi} \otimes mp_{df,k}(S)^{\varphi}$ either a complete canonical representative or a compact canonical representative, which represents the set characterized by $\mathcal{L}_{\beta}(R)^{\varphi} \otimes mp_{df,k}(S)^{\varphi}$. Consequently, service R' is a deadlock-free filtered controller of service S with respect to service R.

In case of responsiveness ($\mathcal{B} = rp_k$), the set of all k-responsive filtered controllers of S with respect to R can be characterized by the product of the annotated structural liberal service automaton $\mathcal{L}_{\gamma}(R)^{\psi^*}$ of service R (representing the set of all services that refines service R under traces), and the responsive operating guideline $mp_{rp,k}(S)^{\psi^*}$ of service S (representing the set of all k-responsive controllers of service S). The correctness of this construction is asserted by the following lemma.

Lemma 7.21 (Characterization of responsive filtered controllers).
For each message bound $k \in \mathbb{N}$ and each two interface compatible services R and S:

$$rp_k\text{-}f\text{-}Controllers(R, S) = Comply_{\gamma}(\mathcal{L}_{\gamma}(R)^{\psi^*} \otimes mp_{rp,k}(S)^{\psi^*}).$$

Proof. $rp_k\text{-}f\text{-}Controllers(R, S) = tr\text{-}refine(R) \cap rp_k Controllers(S)$ [by definition]
$= Comply_{\gamma}(\mathcal{L}_{\gamma}(R)^{\psi^*}) \cap Comply_{\gamma}(mp_{rp,k}(S)^{\psi^*})$ [Lemma 7.18 and Lemma 3.69]
$= Comply_{\gamma}(\mathcal{L}_{\gamma}(R)^{\psi^*} \otimes mp_{rp,k}(S)^{\psi^*})$. [by Corollary 3.21] □

Clearly, an empty set characterized by the synthesized filtering guideline $\mathcal{L}_{\gamma}(R)^{\psi^*} \otimes mp_{rp,k}(S)^{\psi^*}$ means that it is not possible to repair the behavior of service R by removing communication behavior such that the resulting service is a responsive controller of S.

The filtering guideline $\mathcal{L}_{\gamma}(R)^{\psi^*} \otimes mp_{rp,k}(S)^{\psi^*}$ is not a service, but an annotated service automaton representing a finite representation of the set $rp_k\text{-}f\text{-}Controllers(R, S)$. In case the constructed filter is not empty, we can apply similar technique as for deadlock freedom to synthesize R' from the annotated service automaton $\mathcal{L}_{\gamma}(R)^{\psi^*} \otimes mp_{rp,k}(S)^{\psi^*}$. Consequently, service R' is a solution for responsive filtered controller of service S with respect to service R.

The idea of the *filter* operation for services is related to the behavioral constraints on services proposed by Lohmann, Massuthe, and Wolf [2007b] in the sense that all trace-refined services can be expressed as *behavioral constraints*. All trace-refined services that are also substitutes for a given service can be *enforced*, therefore filtering out incorrect behavior, yielding a customized operating guideline which represents the set of all targeted filtered controller.

The related implementation of this procedure is currently under development and will be integrated into in the the open source software tool *Evans* [Parnjai, 2011b].

Filtering Non-Substitute under Inclusion

In this section, we lift up the filtering guidelines technique for Scenario 1 to a filter guidelines technique for Scenario 2. Given a service T which is not a substitute for service S under $\mathcal{B}$ inclusion, we want to filter a service T and derive service T' as an improved

version of T that is a substitute for service S under $\mathcal{B}$ inclusion. A targeted service T' is a service in which the incorrect sequences of communication events and choices have been removed from T. Technically, we can describe the set of all sequences of communication events of a given service T by using its traces and represent a service with less sequences of communication events as a service that refines T under traces (cf. Section 4.2). With a similar idea to the solution for Scenario 1, this allows us to describe the targeted service T' as a service that refines service T under traces (i. e., T' has less or the same traces as T), yet T' is a substitute for service S under $\mathcal{B}$ inclusion.

For this purpose, we define a filtered substitute for service S with respect to service T under $\mathcal{B}$ inclusion as follows.

Definition 7.22 (Filtered substitute under inclusion).
Let $k \in \mathbb{N}$ be a message bound on each message channel. Let S, T and T' be three interface equivalent service automata. Then, service T' is a *filtered substitute* for service S with respect to service T *under* $\mathcal{B}$ *inclusion* iff T' refines T under traces and T' is a substitute for service S under $\mathcal{B}$ inclusion.

The set of all filtered substitutes for service S with respect to service T under $\mathcal{B}$ inclusion is denoted by $\mathcal{B}\text{-}f\text{-}Inclusion(T, S) = tr\text{-}refine(T) \cap \mathcal{B}\text{-}Inclusion(S)$ for $\mathcal{B} \in \{df_k, rp_k\}$.

Clearly, $\mathcal{B}\text{-}f\text{-}Inclusion(T, S)$ is possibly an empty set, meaning that there exists no filter for service S with respect to service T under $\mathcal{B}$ inclusion. The set $\mathcal{B}\text{-}f\text{-}Inclusion(T, S)$ and a targeted service T' are illustrated in Figure 7.23.

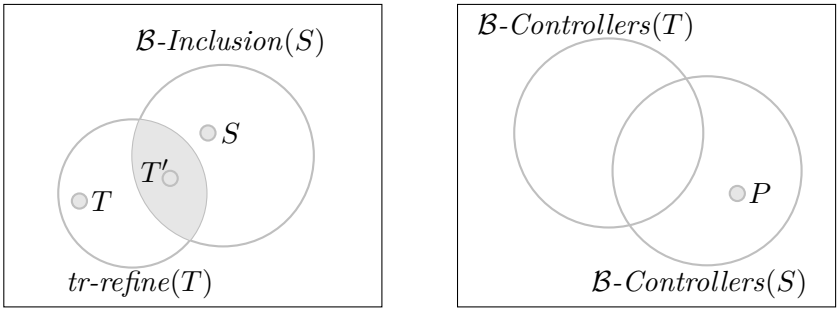


Figure 7.23.: The highlighted intersection represents the set $\mathcal{B}\text{-}f\text{-}Controllers(T, S)$ of all filtered substitutes for service S with respect to service T under $\mathcal{B}$ inclusion.

In case of deadlock freedom ($\mathcal{B} = df_k$), the set of all filtered substitutes for service S with respect to T under deadlock freedom inclusion can be characterized by the product of two annotated service automata; firstly, an annotated strong liberal service automaton $\mathcal{L}_\beta(T)^\varphi$ of service T that represents the set of all services that refine service T under traces, and secondly, the deadlock-free operating guideline $mp_{df,k}(max_{df,k}(S))^\varphi$ of a maximal deadlock-free controller $max_{df,k}(S)$ of service S that represents the set of all substitutes for service S under deadlock freedom inclusion. The correctness of this construction is asserted by the following lemma.

Lemma 7.23 (Characterization of filtered substitutes under deadlock freedom inclusion).
For each message bound $k \in \mathbb{N}$ and each two interface equivalent services S and T:

$$df_k\text{-}f\text{-}Inclusion(T,S) = Comply_\beta(\mathcal{L}_\beta(T)^\varphi \otimes mp_{df,k}(\ max_{df,k}(S)\ )^\varphi).$$

Proof. $df_k\text{-}f\text{-}Inclusion(T,S) = tr\text{-}refine(T) \cap df_k Inclusion(S)$ [by definition]
$= tr\text{-}refine(T) \cap df_k Controllers(\ max_{df,k}(S)\ )$ [Theorem 5.7 in case $\mathcal{B} = df_k$]
$= Comply_\beta(\mathcal{L}_\beta(T)^\varphi) \cap Comply_\beta(mp_{df,k}(\ max_{df,k}(S)\ )^\varphi)$ [Lemma 7.16 and Lemma 3.51]
$= Comply_\beta(\mathcal{L}_\beta(T)^\varphi \otimes mp_{df,k}(S)^\varphi)$. [by Corollary 3.20] □

In case the synthesized filtering guideline $\mathcal{L}_\beta(T)^\varphi \otimes mp_{df,k}(\ max_{df,k}(S)\ )^\varphi$ is empty, it is not possible to repair behavior of service T by removing communication behavior from T such that the resulting service is a substitute for service S under deadlock freedom inclusion. Note that it is possible to employ either a complete canonical deadlock-free controller $\mathcal{C}_{df,k}(S)$ of S or a compact canonical deadlock-free controller $\widehat{\mathcal{C}}_{df,k}(S)$ of S as a solution for $max_{df,k}(S)$ (cf. Section 5.1.2).

The synthesized filtering guideline is not a service, but an annotated service automaton representing a finite representation of the set $df_k\text{-}f\text{-}Inclusion(T,S)$. In case the filtering guideline is not empty, we can apply similar technique as for filtering a non-controller to synthesize T' from the synthesized filtering guideline. Consequently, service T' is a filtered substitute for service S with respect to service T.

In case of responsiveness ($\mathcal{B} = rp_k$), the set of all filtered substitutes for service S with respect to T under responsiveness inclusion can be characterized by the product of two annotated service automata; firstly, an annotated structural liberal service automaton $\mathcal{L}_\gamma(T)^{\psi^*}$ of service T (representing the set of all services that refines service T under traces), and secondly, the responsive operating guideline $mp_{rp,k}(max_{rp,k}(S))^{\psi^*}$ of a maximal responsive controller $max_{rp,k}(S)$ of service S (representing the set of all substitutes for service S under responsiveness inclusion). The correctness of this construction is asserted by the following lemma.

Lemma 7.24 (Characterization of filtered substitutes under responsiveness inclusion).
For each message bound $k \in \mathbb{N}$ and each two interface equivalent services S and T:

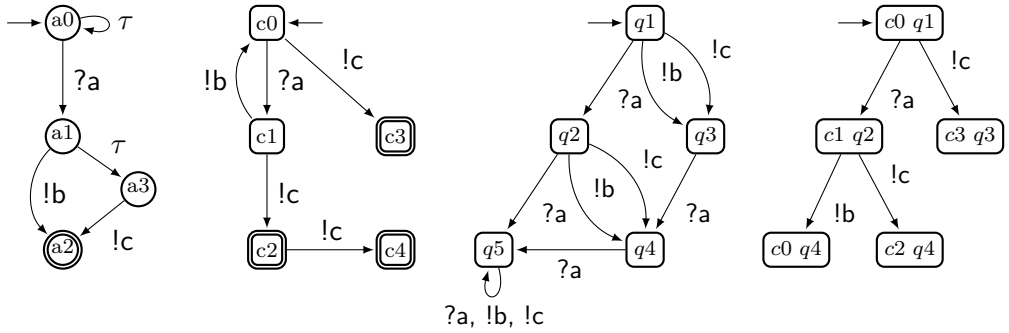
$$rp_k\text{-}f\text{-}Inclusion(T,S) = Comply_\gamma(\mathcal{L}_\gamma(T)^\varphi \otimes mp_{rp,k}(\ max_{rp,k}(S)\ )^{\psi^*}).$$

Proof. $rp_k\text{-}f\text{-}Inclusion(T,S) = tr\text{-}refine(T) \cap rp_k Inclusion(S)$ [by definition]
$= tr\text{-}refine(T) \cap rp_k Controllers(\ max_{rp,k}(S)\ )$ [Theorem 5.7 in case $\mathcal{B} = rp_k$]
$= Comply_\gamma(\mathcal{L}_\gamma(T)^{\psi^*}) \cap Comply_\gamma(mp_{rp,k}(max_{rp,k}(S))^{\psi^*})$ [Lemma 7.18 and Lemma 3.69]
$= Comply_\gamma(\mathcal{L}_\gamma(T)^{\psi^*} \otimes mp_{rp,k}(max_{rp,k}(S))^{\psi^*})$. [by Corollary 3.21] □

In case the synthesized filtering guideline $\mathcal{L}_\gamma(T)^{\psi^*} \otimes mp_{rp,k}(\ max_{rp,k}(S)\ )^{\psi^*}$ is empty, it is not possible to repair the behavior of service T by removing communication behavior

from T such that a repaired service is a substitute for service S under responsiveness inclusion. Note that it is possible to employ either a complete canonical responsive controller $\mathcal{C}_{rp,k}(S)$ of S or a compact canonical responsive controller $\widehat{\mathcal{C}}_{rp,k}(S)$ of S as a solution for $max_{rp,k}(S)$ (cf. Section 5.1.3).

The synthesized filtering guideline is not a service, but an annotated service automaton representing a finite representation of the set rp_k-*f-Inclusion*(T, S). In case the filtering guideline is not empty, we can apply similar technique as for filtering a non-controller to synthesize T' from the synthesized filtering guideline. Consequently, service T' is a solution for a filtered substitute for service S with respect to service T.



(a) U_4 (b) $\mathcal{L}_\gamma(U_3)^{\psi^*}$ (c) *OG* (d) $\mathcal{L}_\gamma(U_3)^{\psi^*} \otimes$ *OG*

$$
\begin{array}{lcl}
\widehat{\psi^*}(c0) & = & \{\ \{?a\}, \{!c\}, \{\ \}\ \} \\
\widehat{\psi^*}(c1) & = & \{\ \{!b\}, \{!c\}, \{\ \}\ \} \\
\widehat{\psi^*}(c2) & = & \{\ \{!c\}, \{\mathit{final}\}, \{\ \}\ \} \\
\widehat{\psi^*}(c3) & = & \{\ \{\mathit{final}\}, \{\ \}\ \} \\
\widehat{\psi^*}(c4) & = & \{\ \{\mathit{final}\}, \{\ \}\ \} \\
 & & \\
\widehat{\psi^*}(q1) & = & \{\ \{?a\}, \{!b,\}, \{!c\}\ \} \\
\widehat{\psi^*}(q2) & = & \{\ \{!b\}\{!c\}\ \} \\
\widehat{\psi^*}(q3) & = & \{\ \{?a\}\ \} \\
\widehat{\psi^*}(q4) & = & \{\ \{\mathit{final}\}\ \} \\
\widehat{\psi^*}(q5) & = & \{\ \{\}, \{?a\}, \{!b\}, \{!c\}, \{\mathit{final}\}\ \} \\
 & & \\
\widehat{\psi^*}(c0\ q1) & = & \{\ \{?a\}, \{!c\}\ \} \\
\widehat{\psi^*}(c1\ q2) & = & \{\ \{!b\}, \{!c\}\ \} \\
\widehat{\psi^*}(c0\ q4) & = & \{\ \{\mathit{final}\}\ \} \\
\widehat{\psi^*}(c2\ q4) & = & \{\ \{\mathit{final}\}\ \} \\
\widehat{\psi^*}(c3\ q3) & = & \{\ \}
\end{array}
$$

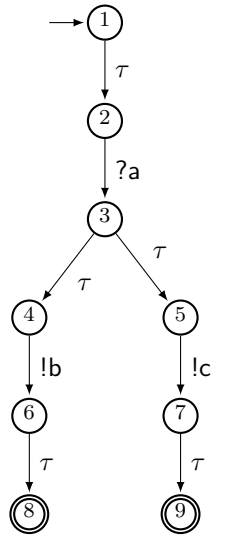


Figure 7.24.: (a) Service U_4 and (d) the product $(\mathcal{L}_\gamma(U_3)^{\psi^*} \otimes OG)$ of $\mathcal{L}_\gamma(U_3)^{\psi^*}$ (from Figure 7.22) and OG where OG is an k-responsive operating guidelines that represents the set rp_k*Inclusion*(U_4) of service U_4, for message bound $k = 1$.

Example 7.25. Consider service U_4 from Figure 7.24. Suppose service U_3 from Figure 7.22 is an upgraded service from U_1. Obviously, service U_3 is not a substitute for service U_4 under k-responsiveness inclusion for message bound $k = 1$ on each message channel.

We apply filtering guidelines technique to construct from the given services U_4 and U_3 a target service that is a substitute for U_4 with respect to U_3 under k-responsiveness inclusion. First, we construct from U_3 its structural liberal service $\mathcal{L}_\gamma(U_3)^{\psi^*}$ representing the set of services that refine U_3 under traces. Next, we construct a k-responsive operating guideline OG of a maximal responsive controller $max_{df,k}(U_4)$ of service U_1 that represents the set of all substitutes for service U_4 under k-responsiveness inclusion. The constructed OG is illustrated in Figure 7.24.

Then, we compute the product $(\mathcal{L}_\gamma(U_3)^{\psi^*} \otimes OG)$ of $\mathcal{L}_\gamma(U_3)^{\psi^*}$ and OG, where OG is k-responsive operating guideline that represents the set $rp_k Inclusion(U_1)$ of service U_4 (from Figure 7.9), illustrated in Figure 7.24 as a finite representation of all filtered substitutes for service U_4 with respect to service U_3 under k-responsiveness inclusion. The bottom right corner of Figure 7.24 shows one service that can be synthesized from the product $(\mathcal{L}_\gamma(U_3)^{\psi^*} \otimes OG)$ using the construction procedure of a compact canonical representative of the set described in Section 3.2.3. The synthesized service is a filtered substitute for service U_4 with respect to service U_3 under k-responsiveness inclusion. $\lhd$

The related implementation of this procedure is currently under development and will be integrated intothe open source software tool *Evans* [Parnjai, 2011b].

7.4.3. Using Filtering Equivalence Guidelines

In this section, we present filtering guidelines as a solution for scenario 3. Given a service T that is not a substitute for service S under $\mathcal{B}$ equivalence. We want to filter a service T and derive service T' as an improved version of T that is a substitute for service S under $\mathcal{B}$ equivalence. A targeted service T' is a service in which the incorrect sequences of communication events and choices have been removed from T. Technically, we can describe the set of all sequences of communication events of a given service T using its traces and represent a service with less sequences of communication events as a service that refines T under traces (cf. Section 4.2). With a similar idea to the solutions for Scenario 1 and Scenario 2, this allows us to describe the targeted service T' as a service that refines service T under traces (i. e., T' has less or the same traces as T), yet T' is a substitute for service S under $\mathcal{B}$ equivalence. Though this technique is restricted to repair only service with deterministic behavior.

For this purpose, we define a filtered substitute for service S with respect to service T under $\mathcal{B}$ equivalence as follows.

Definition 7.26 (Filtered substitute under equivalence).
Let $k \in \mathbb{N}$ be a message bound on each message channel. Let S, T and T' be three interface equivalent service automata. Then, service T' is a *filtered substitute* for service

S with respect to service T *under* $\mathcal{B}$ *equivalence* iff T' refines T under traces and T' is a substitute for service S under $\mathcal{B}$ equivalence.

The set of all filtered substitutes for service S with respect to service T under $\mathcal{B}$ equivalence is denoted by $\mathcal{B}\text{-}f\text{-}Equiv(T, S) = tr\text{-}refine(T) \cap \mathcal{B}\text{-}Equiv(S)$ for $\mathcal{B} \in \{df_k, rp_k\}$.

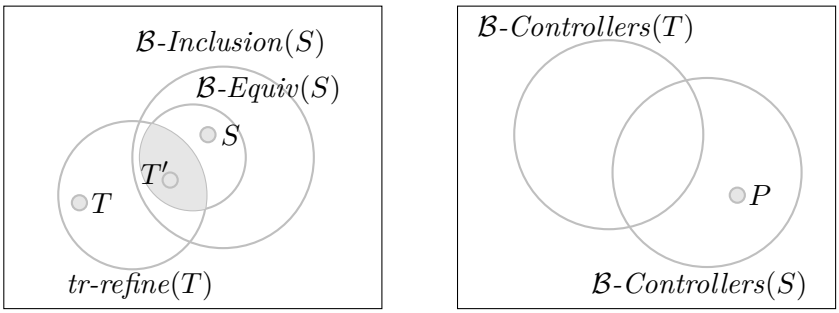


Figure 7.25.: The highlighted intersection represents the set $\mathcal{B}\text{-}f\text{-}Equiv(T, S)$ of all filtered substitutes for service S with respect to service T under $\mathcal{B}$ equivalence.

Clearly, $\mathcal{B}\text{-}f\text{-}Equiv(T, S)$ is possibly an empty set, this means that it is not possible to find a filtered substitute for service of S with respect to service T under $\mathcal{B}$ equivalence. The set $\mathcal{B}\text{-}f\text{-}Equiv(T, S)$ and a targeted service T' are illustrated in Figure 7.25.

Nevertheless, we can not use the same technique as described in Section 7.4.2, because the representation of the set $\mathcal{B}\text{-}Equiv(S)$ is not an annotated service automaton, but an annotated service automaton that is equipped with extra information (cf. an equivalence guideline for S: Section 6.2.4 in case $\mathcal{B} = df_k$ and Section 6.3.4 in case $\mathcal{B} = rp_k$). To find a solution for a filtered substitute T' for service S with respect to service T under $\mathcal{B}$ equivalence, we present a construction procedure of T' from the two given services S and T described by Algorithm 2. The construction requires to construct an equivalence guideline $[\mathcal{C}^2_{\mathcal{B}}(S), Cov(\mathcal{C}^2_{\mathcal{B}}(S))]$ for service S.

Note, that the construction procedure described by Algorithm 2 is quite similar to the procedure described by Algorithm 1 for constructing a minimal public view (cf. Section 7.3.3), only the selected criterion of covering situations are different. This procedure of constructing T' aims to construct a substitute for service S that is not smallest possible in size, but is most similar to T in the sense that the fewest possible communication behavior must be removed from T in order to synthesize T'.

The differences between Algorithm 1 and Algorithm 2 are highlighted as follows. For each $e \in E$, the set of situations of $\mathcal{C}^2_{\mathcal{B}}(S)$ is selected from the set e using the following preference: choose *every* situation $[q_S, \mathcal{M}_S]$ in which there is state q_T of T reachable from its initial state q_{0T} by performing the same trace σ as performed by state q_S of $\mathcal{C}^2_{\mathcal{B}}(S)$ from its initial state q_{0S}. In case the selected set of situations from e is empty, this means, there is no trace of T that matches with a trace reachable at state q_S of $\mathcal{C}^2_{\mathcal{B}}(S)$ such that it can cover a covering situation $[q_S, \mathcal{M}_S]$ in e. In such cases, the procedure

Algorithm 2: construction of a filtered substitute T' for service T w. r. t. service T under $\mathcal{B}$ equivalence

input : (1) $\mathcal{B}$-controllable service S, (2) service T, and (3) message bound k
output : a filtered substitute T' for service T w. r. t. service T under $\mathcal{B}$ equivalence
begin
  compute a complete canonical substitute $\mathcal{C}^2_{\mathcal{B}}(S)$ of S ;
  compute a most permissive controller $\mathcal{B}$-$mp(\mathcal{C}^2_{\mathcal{B}}(S))$ of $\mathcal{C}^2_{\mathcal{B}}(S)$;
  for *each state* q_m *of* $\mathcal{B}$-$mp(\mathcal{C}^2_{\mathcal{B}}(S))$ **do**
    compute a transition system TS of situations of $\mathcal{C}^2_{\mathcal{B}}(S)$ associated with state q_m ;
    compute the partition $E = \xi_{\widehat{\Sigma}(q_m)}\widehat{\mathcal{K}}(q_m)$ at q_m ;
    for *each* e *in* E **do**
      if select(e, T) *is empty* **then** *return empty* T';
      for *each* $[q_S, \mathcal{M}_S]$ *in* select(e, T) **do**
        if q_S *is not marked* **then** *mark*$([q_S,\mathcal{M}_S,], TS)$;
  derive T' from $\mathcal{C}^2_{\mathcal{B}}(S)$ by removing all states that are not marked including their adjacent transitions ;
  if T' *is not empty* **then**
    apply transformation rules $F(1)$, $R(1)$, and $R(2)$ wherever possible on T' to remove τ transitions ;

returns an empty service T', meaning, it is not possible to find a filtered substitute T' for service of S with respect to service T under $\mathcal{B}$ equivalence.

In case the set of selected situations is not empty for every e in the partition $E = \xi_{\widehat{\Sigma}(q_m)}\widehat{\mathcal{K}}(q_m)$ at every state q_m of a most permissive controller $\mathcal{B}$-$mp(\mathcal{C}^2_{\mathcal{B}}(S))$, it is possible to find a filtered substitute T' for service S with respect to service T under $\mathcal{B}$ equivalence by removing as fewe communication behavior from service T as possible. Consequently, the synthesized service T' is a solution for a filtered substitute for service S with respect to service T under $\mathcal{B}$ equivalence for $B \in \{df_k, rp_k\}$.

The implementation of the equivalence guidelines and Algorithm 2 are not yet implemented at the time of writing this thesis.

7.5. Concluding Remarks

In this chapter, we presented techniques that address the *three* problem classes that a service designer, e. g., a domain expert, usually encounters during the design phase when performing service substitution. These techniques provide support for a service designer to *decide* service substitutability, to *synthesize* a service that satisfies behavioral constraints, and to *repair* an incorrect service in a way that every desirable partner of a given service is preserved under substitution criterion.

We sketched various scenarios in which our techniques are applicable, yet, not restricted to, and therefore we illustrated how to apply our results to solve the problems introduced by these scenarios. Our techniques have been developed upon previous work on service substitution and correctness of services and their composition, which are centered around *operating guidelines* for services [Lohmann et al., 2007a, Massuthe, 2009].

The operating guidelines technique has demonstrated its value for characterizing the (possibly infinite) set of all controllers of a given service. This allows a service designer to decide service substitutability under inclusion and equivalence by comparing operating guidelines of two services. Nevertheless, an operating guideline is not a service that is a good candidate for representing the set of controller. As a result, further operation on this service does not always fulfill many synthesis tasks and therefore does not always yield the desired results.

As illustrated in Part II of this thesis, we proposed to synthesize a canonical representative of the set of controllers represented by an operating guideline. We have proved its distinguished properties in Part II. These properties suggest many further operations which validate several analysis and synthesis tasks on service substitution. In this chapter, we demonstrated the value of the canonical representative of the set of controllers in various scenarios. Some techniques are made available by a combination of related existing techniques based on operating guidelines [e. g., Lohmann et al., 2007b, Stahl and Wolf, 2008, Lohmann, 2008a, Stahl and Wolf, 2008, Stahl et al., 2009, Stahl and Wolf, 2009]. This combination allows us to provide solutions to more sophisticated analysis and synthesis tasks on service substitution.

To perform analysis and synthesis tasks as presented in this chapter on industrial services such as WS-BPEL processes, we suggest a service designer to use the compilers and tools provided in the family *service-technology.org*[3]. First, a service designer can translate WS-BPEL processes into our formal model using the compiler *BPEL2oWFN* [Lohmann, 2007] and *PnAPI* [Mennicke et al., 2009].

Under the course of this thesis, we provide the tools *Maxis* [Parnjai, 2011c] and *Evans* [Parnjai, 2011b] to provide supports for analysis and synthesis task related to service substitution. A service designer can use these tools in combination with other tools such as *Fiona* [Massuthe and Weinberg, 2008], *Wendy* [Lohmann and Weinberg, 2010], *Cosme* [Lehmann, 2011], *LoLA* [Schmidt, 2000, Wolf, 2007b], and *Rachel* [Lohmann, 2008b], to perform tasks that have been investigated in this chapter.

For a tool that produces service automaton model that fulfills a synthesis task as an output, a service designer can employ the compiler *PnAPI* and *oWFN2BPEL* [Lohmann and Kleine, 2008] to translate the output service automaton into an abstract *WS-BPEL* process. To this respect, the synthesized service is regarded as a *communication skeleton* which need to be manually refined further. Therefore, a number of tools in the family provide support for closing the gap between our formal model and the real-life services.

[3] illustrated in Figure 2.9 and available at `http://service-technology.org/tools`

Part IV.

Summary

8. Conclusion

In this thesis, we have developed an approach that assists a service designer, e. g., a domain expert, to perform analysis and synthesis tasks on service substitution for two correctness criteria of service composition; (classical) deadlock freedom and responsiveness. The approach guarantees service substitution under two major substitution criteria; equivalence and inclusion substitution. The equivalence criterion guarantees that the original service S and the refined service S' must have exactly the same set of all controllers. The inclusion criterion is more relaxed than equivalence substitution criterion as it guarantees that the set of all controllers of S must be included in the set of all controllers of the refined service S' under substitution. Our approach can also guarantee service substitution under a less restricted criterion than inclusion in which only selected (possibly non-) controllers of service S are preserved under substitution.

8.1. Summary of Contributions

The central idea of the approach is to systematically investigate the relationship between the set of all substitutes for a service and the set of all its controllers via a special class of controllers called *maximal controller*. A maximal controller of service S is a controller of S that has higher order of inclusion preorder than any other controller in the set of all controllers of S, and by definition, it can be substituted by any controller of S in the set. This relationship allows us to underpin a theoretical foundation of our approach.

In this thesis, we presented four main contributions, each for both classical deadlock freedom and responsiveness criteria.

1. We showed that the problem of characterizing the set of all substitutes for a given service S under inclusion can be reduced to the problem of characterizing the set of all controllers of a maximal controller of service S. We proposed a procedure to synthesize a *maximal controller* from a given service S and showed that it is a canonical representative of the set of controllers of S. The synthesis procedure was developed on top of the operating guidelines approach for characterizing the set of all controllers (cf. Chapter 4). We proposed to synthesize an operating guidelines of a synthesized maximal controller of S. Then the problem of deciding whether or not the refined service S' can substitute service S under inclusion can be reduced to the problem of whether or not S' satisfies the matching relation of Operating Guidelines of a maximal controller of S.

2. We proposed a procedure to realize the *closure operation* on service S in which service S is mapped onto its own equivalence class (cf. Chapter 5). The synthesis algorithm

produces from service S one of its equivalent substitute containing a maximum number of traces among all other substitutes for S. The synthesized equivalent substitutes can be regarded as a canonical representative of all substitutes for S.

In case of deadlock freedom, we employed the stable failures refinement known from the literature as a refinement relation of the synthesized equivalent substitutes of S. We showed that a service S' refines the synthesized equivalent substitutes of S under the stable failures whenever S' is a substitute for S under deadlock-free inclusion (cf. Chapter 4 and Chapter 5).

In case of responsiveness, we extended the *stable failures* semantics to a stronger semantics called *responsive failures*. We employed the responsive failures refinement as a refinement relation of the synthesized equivalent substitutes of S and showed that a service S' refines the synthesized equivalent substitutes of S under the responsive failures whenever S' is a substitute for S under responsive inclusion (cf. Chapter 4 and Chapter 5).

3. We proposed the characterization of all equivalent substitutes for a given service (cf. Chapter 6). To decide if the refined service S' is an equivalent substitute for service S, service S' must satisfy two conditions on the synthesized equivalent substitutes for S. Firstly, service S' must refine the synthesized equivalent substitutes for S under the respective failures (stable failures refinement in case of deadlock freedom and responsive failures refinement in case of responsiveness). Secondly, service S' must cover the relevant knowledge that the synthesized equivalent substitutes for S has about all its compatible partners. We showed that service S' satisfies these conditions with respect to service S whenever S' is an equivalent substitute for S under the respective equivalence.

4. We proposed procedures to performing various analysis and synthesis tasks on service substitution (cf. Chapter 7).
   - An alternative procedure to decide service substitutability under deadlock-free inclusion and responsive inclusion.
   - An alternative procedure to synthesize a *public view* of a given service.
   - A procedure to repairing undesirable behavior of the refined service S' with respect to an original service S.
   - A procedure to synthesize substitutes for a service with selected partners.
   - A procedure to integrating selected services for the purpose of finding substitutes for every selected services.
   - A procedure to synthesize substitutes for a given service by means of transformations.

8.2. Open Problems

This section addresses some open problems in thesis.

Repairing Cyclic Services with Undesirable Behavior

When combining with the simulation-based graph edit distance approach [Lohmann, 2008a], we are able to detect and repair an incorrect service that is not a substitute for a given service by performing various edit actions that are necessary to synthesize a substitute that is correct by design. So far the simulation-based graph edit distance approach restricts to a service containing no cycle, there exists no solution for repairing undesirable behavior of cyclic services.

Implementation and Tools Integration of Equivalence Guidelines

The equivalence guideline of a service can be used to repair incorrect behavior of service, though it is not a good candidate for deciding equivalent substitution between two services. Our complexity analysis of the proposed algorithms has shown that the construction of the equivalence guideline involves a number of steps including constructing several operating guidelines as well as exploring state space of the composition between a canonical controller and a canonical substitute for a given service. At the time of writing this thesis, there exists no implementation of the construction of equivalence guidelines for both deadlock freedom and responsiveness. Nevertheless, our experimental results involving some preliminary steps of the construction has shown its applicability to reasonable-sized industrial service models.

Improving Algorithms for Synthesizing Equivalent Substitutes

Our algorithm for constructing a minimal public view followed a simple and naive approach, where the criterion for removing paths from a canonical substitute does not guarantee to deliver always a smallest structure of a minimal public view. To improve a synthesis solution for equivalent substitutes and to battle against the state space explosion problem during synthesis, some reduction techniques (e. g., applying reduction rules proposed by Weinberg [2008]) are required to be used in combination with our equivalence guidelines for synthesizing a minimal equivalent substitute for a given service. An improvement of the algorithm shall also optimize a solution for repairing a non-equivalent substituting service.

8.3. Future Works

This section sketches some ideas for extending results of this thesis.

Other Correctness Criteria of Service Composition

In this thesis, we focused on deadlock freedom and responsiveness as behavioral compatibility criterion for service composition. Neither of them guarantees a termination of service composition, however. One of the elaborated correctness criteria from the literature which guarantees termination is *weak termination* [Stahl, 2009, Massuthe et al.,

2008, Wolf et al., 2011, Weinberg, 2012] requiring that a final state is reachable from any state of the service composition.

To this respect, van Hee, Mooij, Sidorova, and van der Werf [2011] has proposed to extend the concept of maximal controller to weak termination. During the time of writing this thesis, the characterization of all weakly terminated controllers of a given service was not available. Recently, Weinberg [2012] has proposed an operating guideline for weak termination which characterized all weakly terminated controllers of a given service, and Mooij has proposed an extended version of van Hee, Mooij, Sidorova, and van der Werf [2011] for a maximal controller in case of weak termination and that contains in which these results will be soon submitted to a journal.

In computation tree logic (CTL), weak termination can be expressed by $AG.EF\mathit{final}$ in which it is always possible to reach a final state from every state of service composition. It is an interesting direction to extend our data structures to other classes of CTL properties or more general properties.

Services under Multi-ports Setting

In this thesis, we considered services with a single port in the sense of responsiveness defined by Wolf [2009a] and Lohmann [2010]. In case a service is connected to more than one service, it may exchange messages with the other services or the other services may exchange message with each other in an arbitrary fashion. Our responsiveness criterion does not guarantee to respond always to every connected service in the mulltiport setting, such as in the setting of *multiparty contracts* [van der Aalst and Weske, 2001, Leymann et al., 2002, Eshuis and Grefen, 2009, van der Aalst et al., 2008], where several organizations agree beforehand on a common *contract* as a description of process interactions in which the participating organization must conform to.

Service Instance Migration

In this thesis, we did not consider service substitution with running instances. Under the circumstance of long running services, service substitution is not performed under static context. Instead, service substitution is performed while instances of services are still running and communicating with other services. An instance of a service must be migrated to a new instance of the refined service. This problem of instance migration requires the notion of equivalent states between two service instances in order to establish a set of transitions between them. To this direction, Liske [2008] has proposed an algorithm based on operating guidelines approach to solve instance migration problem. Nevertheless, the approach is restricted to deadlock freedom criterion.

Glossary

$\oplus$ ..... composition of service automata
$\otimes$ ..... product of annotated service automata
$\rightarrow$ ..... a transition relation

$\mathcal{B}$ ..... a Beauty predicate of composition
$\mathcal{B}$-*Controllers*(S) the set of all $\mathcal{B}$-controllers of S
$\sqsubseteq_{\mathcal{B}}$ ..... the $\mathcal{B}$ inclusion relation
$=_{\mathcal{B}}$ ..... the $\mathcal{B}$ equivalence relation
$\mathcal{B}$-*Inclusion*(S) . the set of all substitutes for S under $\mathcal{B}$ inclusion
$\mathcal{B}$-*Equiv*(S) ..... the set of all substitutes for S under $\mathcal{B}$ equivalence
$Bags(X)$ ..... a set of all multisets over the set X
$Bags_k(X)$ ..... a set of all k-bounded multisets over the set X
BPEL4Chor .... WS-BPEL extension for choreography modeling [Decker et al., 2008]
BPMN ..... Business Process Modeling Notation [OMG, 2009]

$\mathcal{C}_{df,k}(S)$ ..... a complete canonical k-deadlock-free controller of S
$\widehat{\mathcal{C}}_{df,k}(S)$ ..... a compact canonical k-deadlock-free controller of S
$\mathcal{C}^2_{df,k}(S)$ ..... a complete canonical k-deadlock-free substitute for S
$\widehat{\mathcal{C}}^2_{df,k}(S)$ ..... a compact canonical k-deadlock-free substitute for S
$\mathcal{C}_{rp,k}(S)$ ..... a complete canonical k-responsive controller of S
$\widehat{\mathcal{C}}_{rp,k}(S)$ ..... a compact canonical k-responsive controller of S
$\mathcal{C}^2_{rp,k}(S)$ ..... a complete canonical k-responsive substitute for S
$\widehat{\mathcal{C}}^2_{rp,k}(S)$ ..... a compact canonical k-responsive substitute for S
CCS ..... Calculus of Communicating Systems [Milner, 1989]
$Cov_{\varphi}(S)$ ..... the set of all deadlock-free covering situations of service S
f-*Cover*(S) ..... the set of all service automata that cover the set $Cov_{\varphi}(S)$
$Cov_{\psi}(S)$ ..... the set of all responsive covering situations of service S
rf-cover(S) ..... The set of all service automata that cover the set $Cov_{\psi}(S)$
CSP ..... Communicating Sequential Processes [Hoare, 1985a]

Σ ..... the set of all communicating events
$!\Sigma$ ..... the set of all message sending events
$?\Sigma$ ..... the set of all message receiving events
$\mathbb{E}$ ..... the set of all action events
$\xi_E(\mathcal{K})$ ..... an equivalence partition of $\mathcal{K}$ with respect to E

final ..... a successfully terminating event ($\tau \notin \Sigma$)
rp-failures(S) ... the set of all responsive failures of S

$\sqsubseteq_{RF}$	the responsive failures refinement relation
$=_{RF}$	the responsive failures equivalence relation
rf-refine(S)	the set of all services that refine service S under responsive failures
rf-equiv(S)	the set of all services that are equivalent to service S under responsive failures
failures(S)	the set of all stable failures of S
$\sqsubseteq_{SF}$	the stable failures refinement relation
$=_{SF}$	the stable failures equivalence relation
f-refine(S)	the set of all services that refine service S under stable failures
f-equiv(S)	the set of all services that are equivalent to service S under stable failures
I	the input message channel of a service automaton
k	a message bound $k \in \mathbb{N}$ on each message channel
df_k	the k-bounded deadlock-free compatibility criterion
rp_k	the k-bounded responsive compatibility criterion
$\mathbb{M}$	the set of all message channels
$\mathcal{B}max(S)$	a maximal $\mathcal{B}$ controller of S
$\mathcal{B}min(S)$	a minimal $\mathcal{B}$-controller of S
$\mathbb{M}^I$	the set of all input message channels
$\mathbb{M}^O$	the set of all output message channels
$mp_{df,k}(S)$	a most permissive k-deadlock-free controller of S
$mp_{rp,k}(S)$	a most permissive k-responsive controller of S
O	the output message channel of a service automaton
Q	a set of states
q	a state ($q \in Q$)
q_0	the initial state ($q_0 \in Q$)
S	a service automaton
SOA	Service-Oriented Architecture
SOC	Service-Oriented Computing
τ	an non-communicating event ($\tau \notin \Sigma$)
traces(S)	the set of all traces of S
$\sqsubseteq_{tr}$	the trace refinement relation
$=_{tr}$	the trace equivalence relation
tr-refine(S)	the set of all services that refine service S under traces
tr-equiv(S)	the set of all services that are equivalent to service S under traces
ϱ	a strong simulation or structural matching relation
WS-BPEL	Web Service Business Process Execution Language [Alves et al., 2007]
WS-CDL	Web Service Choreography Description Language [Kavantzas et al., 2005]
WSDL	Web Service Description Language [Chinnici et al., 2003]

Bibliography

A. Alves et al. Web Services Business Process Execution Language Version 2.0. OASIS Standard, OASIS, April 2007. URL `http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.pdf`. (cited at pp. 3, 38, 44, 99, 172, and 274)

R. Backhouse. Galois connections and fixed point calculus. In R. Backhouse, R. Crole, and J. Gibbons, editors, *Algebraic and coalgebraic methods in the mathematics of program construction*, volume 2297 of *LNCS*, pages 89–148. Springer, New York, NY, USA, 2002. ISBN 3-540-43613-8. doi:10.1007/3-540-47797-7. (cited at p. 147)

J. C. M. Baeten. A brief history of process algebra. *Theoretical Computer Science*, 335 (2-3):131–146, May 2005. doi:10.1016/j.tcs.2004.07.036. (cited at pp. 38, 41, and 43)

M. Baldoni, C. Baroglio, A. Martelli, and V. Patti. A priori conformance verification for guaranteeing interoperability in open environments. In A. Dan and W. Lamersdorf, editors, *Service-Oriented Computing - ICSOC 2006*, volume 4294 of *LNCS*, pages 339–351. Springer, Berlin / Heidelberg, 2006. ISBN 978-3-540-68147-2. doi:10.1007/11948148_28. (cited at pp. 15 and 38)

J. A. Bergstra, A. Ponse, and S. A. Smolka, editors. *Handbook of Process Algebra*. Elsevier Science Inc., New York, NY, USA, 2001. ISBN 978-0-444-82830-9. (cited at pp. 38, 41, and 43)

G. V. Bochmann and C. A. Sunshine. Formal methods in communication protocol design. *IEEE Transactions on Communications*, 28(4):624–631, 1980. doi:10.1109/TCOM.1980.1094685. (cited at p. 38)

L. Bordeaux, G. Salaün, D. Berardi, and M. Mecella. When are two web services compatible? In M.-C. Shan, U. Dayal, and M. Hsu, editors, *Technologies for E-Services*, volume 3324 of *LNCS*, pages 15–28. Springer, Berlin / Heidelberg, 2005. ISBN 978-3-540-25049-4. doi:10.1007/978-3-540-31811-8_2. (cited at pp. 4 and 39)

M. Bravetti and G. Zavattaro. A theory for strong service compliance. In *Proceedings of the 9th international conference on Coordination models and languages*, COORDINATION'07, pages 96–112, Berlin, Heidelberg, 2007. Springer. ISBN 978-3-540-72793-4. doi:10.1007/978-3-540-72794-1_6. (cited at p. 41)

J. Bretschneider. Produktbedienungsanleitungen zur Charakterisierung austauschbarer Services. Diplomarbeit, Humboldt-Universität zu Berlin, March 2007. (cited at pp. 55 and 241)

E. Brinksma, A. Rensink, and W. Vogler. Fair testing. In I. Lee and S. Smolka, editors, *CONCUR '95: Concurrency Theory*, volume 962 of *LNCS*, pages 313–327, Berlin / Heidelberg, 1995. Springer. ISBN 978-3-540-60218-7. doi:10.1007/3-540-60218-6_23. (cited at p. 41)

S. D. Brookes, C. A. R. Hoare, and A. W. Roscoe. A theory of communicating sequential processes. *Journal of the ACM*, 31:560–599, June 1984. ISSN 0004-5411. doi:10.1145/828.833. (cited at pp. 41, 105, and 143)

M. Browne, E. Clarke, and O. Grümberg. Characterizing kripke structures in temporal logic. In H. Ehrig, R. Kowalski, G. Levi, and U. Montanari, editors, *TAPSOFT '87*, volume 249 of *LNCS*, pages 256–270. Springer, Berlin / Heidelberg, 1987. ISBN 978-3-540-17660-2. doi:10.1007/3-540-17660-8_60. (cited at p. 47)

C. G. Cassandras and S. Lafortune. *Introduction to Discrete Event Systems*. Springer, Secaucus, NJ, USA, 2008. ISBN 978-0-387-33332-8. (cited at pp. 29 and 40)

G. Castagna, N. Gesbert, and L. Padovani. A theory of contracts for web services. *ACM Transactions on Programming Languages and Systems*, 31:19:1–19:61, July 2009. ISSN 0164-0925. doi:10.1145/1538917.1538920. (cited at p. 41)

R. Chinnici, M. Gudgin, J.-J. Moreau, and S. Weerawarana. Web Services Description Language (WSDL) Version 1.2 Part 1: Core Language. World Wide Web Consortium, Working Draft WD-wsdl12-20030611, June 2003. (cited at pp. 3, 38, and 274)

E. M. Clarke and E. A. Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In *Logic of Programs, Workshop*, pages 52–71, London, UK, 1982. Springer. ISBN 3-540-11212-X. doi:10.1007/978-3-540-69850-0_12. (cited at pp. 28 and 39)

E. M. Clarke and B.-H. Schlingloff. Model checking. In A. Robinson and A. Voronkov, editors, *Handbook of automated reasoning*, pages 1635–1790. Elsevier Science Publishers B. V., Amsterdam, The Netherlands, The Netherlands, 2001. ISBN 0-444-50812-0. URL `http://dl.acm.org/citation.cfm?id=778522.778533`. (cited at p. 5)

E. M. Clarke, O. Grumberg, and D. Peled. *Model checking*. MIT Press, 2001. ISBN 978-0-262-03270-4. (cited at p. 5)

L. de Alfaro and T. A. Henzinger. Interface automata. In *Proceedings of the 8th European software engineering conference held jointly with 9th ACM SIGSOFT international symposium on Foundations of software engineering*, ESEC/FSE-9, pages 109–120, New York, NY, USA, 2001. ACM. ISBN 1-58113-390-1. doi:10.1145/503271.503226. (cited at pp. 15 and 38)

R. De Nicola. Extensional equivalence for transition systems. *Acta Informatica*, 24: 211–237, April 1987. ISSN 0001-5903. doi:10.1007/BF00264365. (cited at p. 41)

G. Decker, O. Kopp, F. Leymann, K. Pfitzner, and M. Weske. Modeling service choreographies using BPMN and BPEL4Chor. In *20th International Conference, CAiSE 2008 Montpellier, France, June 16-20, 2008 Proceedings*, volume 5074 of *LNCS*, pages 79–93, Berlin / Heidelberg, 2008. Springer. doi:10.1007/978-3-540-69534-9_6. (cited at pp. 3, 38, and 273)

V. Deora, J. Shao, W. A. Gray, and N. J. Fiddian. Modelling quality of service in service oriented computing. In *Proceedings of the Second IEEE International Symposium on Service-Oriented System Engineering*, pages 95–101, Washington, DC, USA, 2006. IEEE Computer Society. ISBN 0-7695-2726-4. doi:10.1109/SOSE.2006.22. (cited at pp. 4 and 39)

D. L. Dill. *Trace theory for automatic hierarchical verification of speed-independent circuits*. MIT Press, 1989. ISBN 0-262-04101-4. (cited at pp. 42 and 150)

D. Dill. Complete trace structures. In M. Leeser and G. Brown, editors, *Proceedings of the Mathematical Sciences Institute workshop on Hardware specification, verification and synthesis: mathematical aspects*, volume 408 of *LNCS*, pages 224–243. Springer, Berlin / Heidelberg, 1990. ISBN 978-0-387-97226-8. doi:10.1007/0-387-97226-9_31. (cited at pp. 42 and 150)

X. Dong, A. Halevy, J. Madhavan, E. Nemes, and J. Zhang. Similarity search for web services. In *Proceedings of the Thirteenth international conference on Very large data bases - Volume 30*, VLDB '04, pages 372–383. VLDB Endowment, 2004. ISBN 0-12-088469-0. (cited at pp. 4 and 39)

M. Dumas, B. Benatallah, and H. R. M. Nezhad. Web service protocols: Compatibility and adaptation. *IEEE Data Engineering Bulletin*, pages 40–44, 2008. (cited at pp. 4 and 39)

E. Emerson and J. Y. Halpern. Decision procedures and expressiveness in the temporal logic of branching time. *Journal of Computer and System Sciences*, 30(1):1 – 24, 1985. ISSN 0022-0000. doi:10.1016/0022-0000(85)90001-7. (cited at pp. 28 and 39)

R. Eshuis and P. W. P. J. Grefen. Composing services into structured processes. *International Journal of Cooperative Information Systems*, 18(2):309–337, 2009. doi:10.1142/S0218843009002026. (cited at pp. 42 and 272)

C. Fournet, C. A. R. Hoare, S. K. Rajamani, and J. Rehof. Stuck-free conformance. In *16th International Conference, CAV 2004, Boston, MA, USA, July 13-17, 2004. Proceedings*, volume 3114 of *LNCS*, pages 242–254, Berlin / Heidelberg, 2004. Springer. ISBN 978-3-540-22342-9. doi:10.1007/978-3-540-27813-9_19. (cited at p. 41)

C. Gierds. Strukturelle Reduktion von Bedienungsanleitungen. Diplomarbeit, Humboldt-Universität zu Berlin, January 2008a. (cited at pp. 40 and 223)

C. Gierds. Finding cost-efficient adapters. In N. Lohmann and K. Wolf, editors, *Proceedings of the 15th German Workshop on Algorithms and Tools for Petri Nets, AWPN 2008, Rostock, Germany, September 26–27, 2008*, volume 380 of *CEUR Workshop Proceedings*, pages 37–42. CEUR-WS.org, September 2008b. URL `http://CEUR-WS.org/Vol-380/paper06.pdf`. (cited at p. 254)

C. Gierds and J. Sürmeli. Estimating costs of a service. In C. Gierds and J. Sürmeli, editors, *Proceedings of the 2nd Central-European Workshop on Services and their Composition, ZEUS 2010, Berlin, Germany, February 25–26, 2010*, volume 563 of *CEUR Workshop Proceedings*, pages 121–128. CEUR-WS.org, 2010. URL `http://CEUR-WS.org/Vol-563/paper15.pdf`. (cited at p. 254)

C. Gierds, A. J. Mooij, and K. Wolf. Reducing adapter synthesis to controller synthesis. *IEEE Transactions on Services Computing*, 99(PrePrints), 2010. ISSN 1939-1374. doi:10.1109/TSC.2010.57. (cited at p. 40)

D. Grigori, J. C. Corrales, and M. Bouzeghoub. Behavioral matchmaking for service retrieval: Application to conversation protocols. *Information Systems*, 33(7-8):681–698, 2008. doi:10.1016/j.is.2008.02.004. (cited at p. 38)

J. Hidders, M. Dumas, W. M. P. van der Aalst, A. H. M. ter Hofstede, and J. Verelst. When are two workflows the same? In *Proceedings of the 2005 Australasian symposium on Theory of computing - Volume 41*, CATS '05, pages 3–11, Darlinghurst, Australia, 2005. Australian Computer Society, Inc. ISBN 1-920682-23-6. (cited at p. 41)

S. Hinz, K. Schmidt, and C. Stahl. Transforming BPEL to Petri Nets. In W. M. P. v. d. Aalst, B. Benatallah, F. Casati, and F. Curbera, editors, *Proceedings of the Third International Conference on Business Process Management (BPM 2005)*, volume 3649 of *LNCS*, pages 220–235, Nancy, France, September 2005. Springer. doi:10.1007/11538394_15. (cited at p. 38)

C. A. R. Hoare. Communicating sequential processes. *Communications of the ACM*, 21: 666–677, August 1978. ISSN 0001-0782. doi:10.1145/359576.359585. (cited at pp. 41, 100, 105, and 143)

C. A. R. Hoare. *Communicating sequential processes*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1985a. ISBN 0-13-153271-5. (cited at p. 273)

C. A. R. Hoare. *Communicating sequential processes*. Prentice-Hall International series in computing science. Prentice-Hall International, 1985b. (cited at pp. 9, 39, 41, 105, and 143)

K. Honda. Types for Dyadic Interaction. In *CONCUR*, volume 715 of *LNCS*, pages 509–523. Springer, 1993. (cited at p. 42)

J. E. Hopcroft, R. Motwani, and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, November 2000. ISBN 978-0-2014-4124-6. (cited at p. 38)

K. Kaschner. Managing test suites for services. In M. Schwarick and M. Heiner, editors, *Proceedings of the 17th German Workshop on Algorithms and Tools for Petri Nets (AWPN 2010), Cottbus, Germany, October 7-8, 2010*, volume 643 of *CEUR Workshop Proceedings*, pages 142–147. CEUR-WS.org, October 2010. URL `http://ceur-ws.org/Vol-643/paper17.pdf`. (cited at p. 42)

K. Kaschner. Conformance testing for asynchronously communicating services. In G. Kappel, Z. Maamar, and H. Motahari-Nezhad, editors, *Service-Oriented Computing*, volume 7084 of *LNCS*, pages 108–124. Springer, Berlin / Heidelberg, 2011. ISBN 978-3-642-25534-2. doi:10.1007/978-3-642-25535-9_8. (cited at pp. 42 and 144)

K. Kaschner and K. Wolf. Set algebra for service behavior: Applications and constructions. In U. Dayal, J. Eder, J. Koehler, and H. Reijers, editors, *Business Process Management, 7th International Conference, BPM 2009, Ulm, Germany, September 8-10, 2009, Proceedings*, volume 5701 of *LNCS*, pages 193–210. Springer, September 2009. doi: 10.1007/978-3-642-03848-8_14. (cited at pp. 42, 46, and 241)

K. Kaschner, P. Massuthe, and K. Wolf. Symbolic representation of operating guidelines for services. *Petri Net Newsletter*, 72:21–28, April 2007. (cited at pp. 40 and 223)

N. Kavantzas, D. Burdett, G. Ritzinger, and Y. Lafon. Web Services Choreography Description Language Version 1.0. W3C Candidate Recommendation, W3C, November 2005. URL `http://www.w3.org/TR/ws-cdl-10`. (cited at p. 274)

L. Klensin. Simple Mail Transport Protocol, RFC 2821. Technical report, IETF, April 2001. URL `http://www.apps.ietf.org/rfc/rfc2821.html`. (cited at pp. 99 and 172)

D. König, N. Lohmann, S. Moser, C. Stahl, and K. Wolf. Extending the compatibility notion for abstract ws-bpel processes. In *Proceedings of the 17th international conference on World Wide Web*, WWW '08, pages 785–794, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-085-2. doi:10.1145/1367497.1367603. (cited at pp. 42, 144, and 232)

C. Laneve and L. Padovani. The must preorder revisited. In *18th International Conference, CONCUR 2007, Lisbon, Portugal, September 3-8, 2007. Proceedings*, volume 4703 of *LNCS*, pages 212–225, Berlin / Heidelberg, 2007. Springer. ISBN 978-3-540-74406-1. doi:10.1007/978-3-540-74407-8_15. (cited at p. 41)

P. Laufer. Public-View-Generierung. Diplomarbeit, Humboldt-Universität zu Berlin, November 2007. (cited at p. 236)

A. Lehmann. Cosme: a tool to decide substitutability and equivalence of services using an efficient bit representation of operating guidelines. Tool available at `http://service-technology.org/cosme`., 2011. (cited at pp. 11, 43, 223, and 266)

F. Leymann, D. Roller, and M. Schmidt. Web services and business process management. *IBM Systems Journal*, 41(2):198–211, April 2002. doi:10.1147/sj.412.0198. (cited at pp. 42, 234, and 272)

B. Lin, G. de Jong, and T. Kolks. Modeling and optimization of hierarchical synchronous circuits. In *Proceedings of the 1995 European conference on Design and Test*, EDTC '95, Washington, DC, USA, 1995a. IEEE Computer Society. ISBN 0-8186-7039-8. doi:10.1109/EDTC.1995.470406. (cited at pp. 42 and 150)

B. Lin, G. de Jong, and T. Kolks. Hierarchical optimization of asynchronous circuits. In *Proceedings of the 32nd annual ACM/IEEE Design Automation Conference*, DAC '95, pages 712–717, New York, NY, USA, 1995b. ACM. ISBN 0-89791-725-1. doi: 10.1145/217474.217616. (cited at pp. 42 and 150)

N. Liske. Laufzeitersetzung offener Workflownetze. Diplomarbeit, Humboldt-Universität zu Berlin, July 2008. (cited at p. 272)

N. Lohmann, P. Massuthe, and K. Wolf. Operating guidelines for finite-state services. In *28th International Conference on Applications and Theory of Petri Nets and Other Models of Concurrency, ICATPN 2007, Siedlce, Poland, June 25-29, 2007. Proceedings*, volume 4546 of *LNCS*, pages 321–341, Berlin / Heidelberg, 2007a. Springer. doi: 10.1007/978-3-540-73094-1_20. (cited at pp. 4, 8, 11, 24, 25, 29, 39, 40, 42, 44, 46, 47, 48, 54, 64, 65, 67, 68, 69, 70, 72, 83, 88, 92, 160, 241, 255, and 266)

N. Lohmann. A feature-complete Petri net semantics for WS-BPEL 2.0 and its compiler BPEL2oWFN. Informatik-Berichte 212, Humboldt-Universität zu Berlin, Berlin, Germany, August 2007. Tool available at `http://service-technology.org/bpel2owfn`. (cited at pp. 11, 43, 99, 172, 223, 231, and 266)

N. Lohmann. Correcting deadlocking service choreographies using a simulation-based graph edit distance. In M. Dumas and M. Reichert, editors, *6th International Conference, BPM 2008, Milan, Italy, September 2-4, 2008. Proceedings*, volume 5240 of *LNCS*, pages 132–147, Berlin / Heidelberg, September 2008a. Springer. ISBN 978-3-540-85757-0. doi:10.1007/978-3-540-85758-7_12. (cited at pp. 11, 252, 253, 266, and 271)

N. Lohmann. Rachel: a tool to correct service choreographies. Tool available at `http://service-technology.org/rachel`., 2008b. (cited at pp. 11, 43, and 266)

N. Lohmann. *Correctness of services and their composition*. Dissertation, Universität Rostock / Technische Universiteit Eindhoven, Rostock, Germany / Eindhoven, The Netherlands, September 2010. ISBN 9789038623184. (cited at pp. 8, 22, 25, 26, 29, 39, 40, 46, 47, 48, 55, 76, 83, 88, 160, 173, and 272)

N. Lohmann and J. Kleine. Fully-automatic translation of open workflow net models into simple abstract BPEL processes. In *Modellierung 2008, 12.–14. März 2008, Berlin, Proceeding*, volume P-127 of *Lecture Notes in Informatics (LNI)*, pages 57–72. GI, March 2008. URL `http://wwwteo.informatik.uni-rostock.de/ls_tpp/publications/LohmannK_2008_mod.pdf`. (cited at pp. 38, 43, 231, and 266)

N. Lohmann and D. Weinberg. Wendy: A tool to synthesize partners for services. In *PETRI NETS 2010*, LNCS 6128, pages 297–307. Springer, 2010. Tool available at `http://service-technology.org/wendy`. (cited at pp. 11, 43, 99, 172, 223, and 266)

N. Lohmann and K. Wolf. Petrifying operating guidelines for services. In *Ninth International Conference on Application of Concurrency to System Design (ACSD 2009), 1-3 July 2009, Augsburg, Germany*, pages 80–88. IEEE Computer Society, June 2009. doi:10.1109/ACSD.2009.11. (cited at pp. 40 and 223)

N. Lohmann and K. Wolf. How to implement a theory of correctness in the area of business processes and services. In R. Hull, J. Mendling, and S. Tai, editors, *Business Process Management, 8th International Conference, BPM 2010, Hoboken, NJ, USA, September 14–16, 2010, Proceedings*, volume 6336 of *LNCS*, pages 61–77. Springer, September 2010. doi:10.1007/978-3-642-15618-2_7. (cited at p. 42)

N. Lohmann and K. Wolf. Data under control. In *Proceedings of the 18th German Workshop on Algorithms and Tools for Petri Nets (AWPN 2011), Hagen, Germany, September 29-30, 2011*, September 2011a. (cited at p. 40)

N. Lohmann and K. Wolf. Compact representations and efficient algorithms for operating guidelines. *Fundamenta Informaticae*, 108(1-2):43–62, 2011b. (cited at pp. 40, 144, and 223)

N. Lohmann, P. Massuthe, and K. Wolf. Behavioral constraints for services. In G. Alonso, P. Dadam, and M. Rosemann, editors, *Business Process Management, 5th International Conference, BPM 2007, Brisbane, Australia, September 24-28, 2007, Proceedings*, volume 4714 of *LNCS*, pages 271–287. Springer, September 2007b. doi:10.1007/978-3-540-75183-0_20. (cited at pp. 11, 40, 231, 233, 259, and 266)

N. Lohmann, H. Verbeek, and R. Dijkman. Petri net transformations for business processes – a survey. In *Special Issue on Concurrency in Process-Aware Information Systems*, volume 5460 of *LNCS*, pages 46–63, March 2009a. ISBN 978-3-642-00898-6. doi:10.1007/978-3-642-00899-3_3. (cited at pp. 39 and 44)

N. Lohmann, H. Verbeek, C. Ouyang, and C. Stahl. Comparing and evaluating Petri net semantics for BPEL. *International Journal of Business Process Integration and Management*, 4(1):60–73, 2009b. doi:10.1504/IJBPIM.2009.026986. (cited at pp. 39 and 44)

N. A. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1996. ISBN 1558603484. (cited at pp. 15 and 38)

N. A. Lynch and M. R. Tuttle. An introduction to input/output automata. *CWI Quarterly*, 2:219–246, 1989. (cited at pp. 15 and 38)

P. Massuthe. *Operating Guidelines for Services*. Dissertation, Humboldt-Universität zu Berlin; Eindhoven University of Technology, April 2009. ISBN 978-90-386-1702-2. (cited at pp. 8, 24, 25, 29, 38, 39, 40, 44, 46, 47, 48, 54, 72, 83, 88, 92, 160, 173, and 266)

P. Massuthe and K. Schmidt. Operating Guidelines - an automata-theoretic foundation for the service-oriented architecture. In K.-Y. Cai, A. Ohnishi, and M. Lau, editors, *Proceedings of the Fifth International Conference on Quality Software (QSIC 2005)*, pages 452–457, Melbourne, Australia, September 2005. IEEE Computer Society. doi: 10.1109/QSIC.2005.47. (cited at pp. 8, 15, 38, 40, 44, 92, and 160)

P. Massuthe and D. Weinberg. FIONA: A tool to analyze interacting open nets. In *AWPN 2008*, CEUR Workshop Proceedings Vol. 380, pages 99–104. CEUR-WS.org, September 2008. Tool available at `http://service-technology.org/fiona`. (cited at pp. 11, 43, 223, and 266)

P. Massuthe, W. Reisig, and K. Schmidt. An Operating Guideline approach to the SOA. *Annals of Mathematics, Computing & Teleinformatics*, 1(3):35–43, 2005. (cited at pp. 8, 38, 40, and 44)

P. Massuthe, A. Serebrenik, N. Sidorova, and K. Wolf. Can I find a partner? Undecidablity of partner existence for open nets. *Information Processing Letters*, 108(6):374–378, November 2008. (cited at pp. 28, 39, and 271)

S. Mennicke and N. Lohmann. Rebecca: a tool to realize service choreographies. Tool available at `http://service-technology.org/rebecca`., 2009. (cited at p. 253)

S. Mennicke, C. Sura, R. Waltemath, N. Lohmann, C. Gierds, and M. Znamirowski. Petri net api: a tool to encapsulating petri net-related functions in a c++ api. Tool available at `http://service-technology.org/pnapi`., 2009. (cited at pp. 11, 43, 100, 172, 223, 231, and 266)

P. M. Merlin. Specification and validation of protocols. *IEEE Transactions on Communications*, 27(11):1671–1680, 1979. doi:10.1109/TCOM.1979.1094323. (cited at p. 38)

G. D. Micheli. *Synthesis and Optimization of Digital Circuits*. McGraw-Hill Higher Education, 1st edition, 1994. ISBN 0070163332. (cited at p. 61)

R. Milner. *Communication and concurrency*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1989. ISBN 0-13-115007-3. (cited at pp. 47 and 273)

R. Milner. An algebraic definition of simulation between programs. Technical report, Stanford University, Stanford, CA, USA, 1971. (cited at p. 47)

R. Milner. Calculi for synchrony and asynchrony. *Theoretical Computer Science*, 25(3): 267–310, 1983. doi:10.1016/0304-3975(83)90114-7. (cited at p. 41)

R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, parts I and II. *Information and Computation*, 100:1–77, September 1992. ISSN 0890-5401. doi: 10.1016/0890-5401(92)90008-4. (cited at p. 42)

A. Mooij, J. Parnjai, C. Stahl, and M. Voorhoeve. Constructing replaceable services using operating guidelines and maximal controllers. In M. Bravetti and T. Bultan, editors, *Web Services and Formal Methods*, volume 6551 of *LNCS*, pages 116–130, Berlin / Heidelberg, 2011. Springer. ISBN 978-3-642-19588-4. doi:10.1007/978-3-642-19589-1_8. (cited at pp. 33, 34, 39, 40, and 146)

A. J. Mooij and M. Voorhoeve. Proof techniques for adapter generation. In *5th International Workshop, WS-FM 2008, Milan, Italy, September 4-5, 2008*, volume 5387 of *LNCS*, pages 207–223. Springer, 2009. ISBN 978-3-642-01363-8. doi: 10.1007/978-3-642-01364-5_13. (cited at pp. 33, 39, 40, 92, 146, and 147)

A. J. Mooij, C. Stahl, and M. Voorhoeve. Relating fair testing and accordance for service replaceability. *Journal of Logic and Algebraic Programming*, 79(3–5):233–244, April 2010. doi:10.1016/j.jlap.2009.12.001. (cited at pp. 22, 38, and 41)

M. Mrissa, C. Ghedira, D. Benslimane, Z. Maamar, F. Rosenberg, and S. Dustdar. A context-based mediation approach to compose semantic web services. *ACM Trans. Internet Technol.*, 8, November 2007. ISSN 1533-5399. doi:10.1145/1294148.1294152. (cited at pp. 4 and 39)

R. Müller. On the notion of deadlocks in open nets. In M. Schwarick and M. Heiner, editors, *Proceedings of the 17th German Workshop on Algorithms and Tools for Petri Nets, AWPN 2010, Cottbus, Germany, October 07-08, 2010*, volume 643 of *CEUR Workshop Proceedings*, pages 130–135. CEUR-WS.org, October 2010a. URL `http://ceur-ws.org/Vol-643/paper15.pdf`. (cited at p. 39)

R. Müller. Formal characterisation of partners of an open net. Diplomarbeit, Humboldt-Universität zu Berlin, June 2010b. (cited at p. 39)

T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, April 1989. (cited at pp. 42, 127, 134, and 232)

G. J. Myers and C. Sandler. *The Art of Software Testing*. John Wiley & Sons, 2004. ISBN 0471469122. (cited at pp. 5 and 42)

OMG. Business Process Model and Notation (BPMN). FTF Beta 1 for Version 2.0, Object Management Group, 2009. URL `http://www.omg.org/spec/BPMN/2.0`. (cited at pp. 3, 38, 44, and 273)

O. Ore. Galois connexions. *Transactions of the American Mathematical Society*, 55(3): pp. 493–513, 1944. ISSN 00029947. (cited at p. 147)

M. P. Papazoglou. *Web Services: Principles and Technology*. Pearson, Prentice Hall, 2008a. ISBN 9780321155559. (cited at pp. 3, 4, 38, and 39)

M. P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann. Service-oriented computing: State of the art and research challenges. *Computer*, 40:38–45, November 2007. ISSN 0018-9162. doi:10.1109/MC.2007.400. (cited at pp. 3, 37, and 38)

M. P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann. Service-Oriented computing: A research roadmap. *International Journal of Cooperative Information Systems*, 17 (02):223, 2008. ISSN 0218-8430. doi:10.1142/S0218843008001816. (cited at p. 38)

M. Papazoglou. The challenges of service evolution. In Z. Bellahsène and M. Léonard, editors, *Advanced Information Systems Engineering*, volume 5074 of *LNCS*, pages 1–15. Springer, Berlin / Heidelberg, 2008b. ISBN 978-3-540-69533-2. doi:10.1007/978-3-540-69534-9_1. (cited at pp. 4, 38, and 40)

M. P. Papazoglou. Agent-oriented technology in support of e-business. *Communications of the ACM*, 44(4):71–77, 2001. doi:10.1145/367211.367268. (cited at p. 37)

M. P. Papazoglou. Service -oriented computing: Concepts, characteristics and directions. In *Proceedings of the Fourth International Conference on Web Information Systems Engineering*, WISE '03, Washington, DC, USA, 2003. IEEE Computer Society. ISBN 0-7695-1999-7. (cited at pp. 3, 37, and 38)

D. Park. Concurrency and automata on infinite sequences. In P. Deussen, editor, *Theoretical Computer Science*, volume 104 of *LNCS*, pages 167–183. Springer, Berlin, Heidelberg, 1981. ISBN 978-3-540-10576-3. doi:10.1007/BFb0017309. (cited at p. 41)

J. Parnjai. Filtering undesirable service substitution behaviors using filtering guidelines. In D. Eichhorn, A. Koschmider, and H. Zhang, editors, *Proceedings of the 3rd Central-European Workshop on Services and their Composition, ZEUS 2011, Karlsruhe, Germany, February 21–22, 2011*, volume 705 of *CEUR Workshop Proceedings*, pages 50–57. CEUR-WS.org, 2011a. URL `http://CEUR-WS.org/Vol-705/paper7.pdf`. (cited at p. 254)

J. Parnjai. Evans: a tool to synthesize public view and substitutable services. Tool available at `http://service-technology.org/evans`., 2011b. (cited at pp. 11, 43, 174, 226, 228, 236, 245, 249, 253, 259, 263, and 266)

J. Parnjai. Maxis: a tool to construct a maximal partner of services. Tool available at `http://service-technology.org/maxis`., 2011c. (cited at pp. 11, 43, 99, 172, 225, and 266)

J. Parnjai, C. Stahl, and K. Wolf. A finite representation of all substitutable services and its applications. In *ZEUS 2009*, pages 29–34. CEUR vol. 438, March 2009. URL `http://CEUR-WS.org/Vol-438/paper5.pdf`. (cited at pp. 33, 34, 39, and 40)

C. Peltz. Web services orchestration and choreography. *Computer*, 36:46–52, October 2003. ISSN 0018-9162. doi:10.1109/MC.2003.1236471. (cited at pp. 3 and 37)

P. Ramadge and W. Wonham. Supervisory control of a class of discrete event processes. *Siam Journal on Control and Optimization*, 25(1), 1987. doi:10.1137/0325013. (cited at pp. 29 and 40)

J. Reed, A. W. Roscoe, and J. Sinclair. Responsiveness and stable revivals. *Formal Aspects of Computing*, 19:303–319, 2007. ISSN 0934-5043. doi:10.1007/s00165-007-0032-9. (cited at p. 41)

W. Reisig. *Petri Nets*. Springer, EATCS Monographs on Theoretical Computer Science edition, 1985. (cited at pp. 38 and 43)

W. Reisig. Towards a theory of services. In R. Kaschek, C. Kop, C. Steinberger, G. Fliedl, W. Aalst, J. Mylopoulos, M. Rosemann, M. J. Shaw, and C. Szyperski, editors, *Information Systems and e-Business Technologies*, volume 5 of *Lecture Notes in Business Information Processing*, pages 271–281. Springer, Berlin / Heidelberg, 2008. ISBN 978-3-540-78942-0. doi:10.1007/978-3-540-78942-0_27. (cited at pp. 23 and 38)

W. Reisig, K. Wolf, J. Bretschneider, K. Kaschner, N. Lohmann, P. Massuthe, and C. Stahl. Challenges in a service-oriented world. *ERCIM News*, 70:28–29, July 2007. (cited at p. 38)

A. W. Roscoe. *The theory and practice of concurrency*. Prentice Hall, Upper Saddle River, NJ, USA, 1998. ISBN 0-13-6774409-5. (cited at pp. 9, 41, 100, 105, and 143)

A. Roscoe. Revivals, stuckness and the hierarchy of CSP models. *Journal of Logic and Algebraic Programming*, 78(3):163 – 190, 2009. ISSN 1567-8326. doi:10.1016/j.jlap.2008.10.002. (cited at p. 41)

K. Schmidt. LoLA: A low level analyser. In *Application and Theory of Petri Nets 2000*, volume 1825 of *LNCS*, pages 465–474, Berlin / Heidelberg, June 2000. Springer. doi:10.1007/3-540-44988-4_27. (cited at pp. 43, 225, and 266)

C. Stahl. *Service Substitution - A Behavioral Approach Based on Petri Nets*. Dissertation, Humboldt-Universität zu Berlin, December 2009. ISBN 978-90-386-2065-7. (cited at pp. 9, 22, 28, 29, 33, 34, 38, 39, 40, 41, 42, 46, 47, 48, 54, 55, 56, 74, 88, 110, 144, 173, 241, and 271)

C. Stahl and W. Vogler. A trace-based view on operating guidelines. In M. Hofmann, editor, *Foundations of Software Science and Computational Structures*, volume 6604 of *LNCS*, pages 411–425. Springer, Berlin / Heidelberg, 2011. ISBN 978-3-642-19804-5. doi:10.1007/978-3-642-19805-2_28. (cited at pp. 10, 39, 42, and 144)

C. Stahl and K. Wolf. Covering places and transitions in open nets. In M. Dumas and M. Reichert, editors, *Business Process Management, 6th International Conference, BPM 2008, Milan, Italy, September 1-4, 2008, Proceedings*, volume 5240 of *LNCS*, pages 116–131. Springer, September 2008. doi:10.1007/978-3-540-85758-7_11. (cited at pp. 40, 231, 233, and 266)

C. Stahl and K. Wolf. Deciding service composition and substitutability using extended operating guidelines. *Data & Knowledge Engineering*, 68(9):819–833, 2009. doi:10.1016/j.datak.2009.02.012. (cited at pp. 10, 40, 146, and 266)

C. Stahl, P. Massuthe, and J. Bretschneider. Deciding substitutability of services with Operating Guidelines. *Transactions on Petri Nets and Other Models of Concurrency II*, 2(5460):172–191, March 2009. doi:10.1007/978-3-642-00899-3_10. (cited at pp. 11, 33, 34, 39, 40, 55, 56, 70, 74, 146, 222, 228, 231, 233, 241, 255, and 266)

C. Sura. Sayo: a tool to generate public view service automata from bit operating guidelines. Tool available at `http://service-technology.org/sayo`., 2009a. (cited at p. 236)

C. Sura. The compilers: tools that compile several service-technology format together. Tool available at `http://service-technology.org/sayo`., 2009b. (cited at p. 223)

K. Sycara, M. Paolucci, A. Ankolekar, and N. Srinivasan. Automated discovery, interaction and composition of semantic web services. *Web Semantics: Science, Services and Agents on the World Wide Web*, 1(1), 2011. ISSN 1570-8268. (cited at pp. 4 and 39)

P. Traverso and M. Pistore. Automated composition of semantic web services into executable processes. In *Third International Semantic Web Conference – ISWC 2004, Hiroshima, Japan, November 7-11, 2004. Proceedings*, pages 380–394. Springer, Berlin / Heidelberg, 2004. ISBN 978-3-540-23798-3. doi:10.1007/978-3-540-30475-3_27. (cited at pp. 4 and 39)

G. J. Tretmans. Test generation with inputs, outputs and repetitive quiescence, 1996. URL `http://doc.utwente.nl/65463/`. (cited at pp. 15, 38, 39, and 42)

G. Tretmans. Repetitive quiescence in implementation and testing (extended abstract). In A. Wolisz, I. Schieferdecker, and A. Rennoch, editors, *Formale Beschreibungstechniken für verteilte Systeme*, volume 315 of *GMD-Studien*, pages 23–37. GMD-Forschungszentrum Informationstechnik GmbH, June 1997. URL `http://doc.utwente.nl/63309/`. (cited at p. 38)

A. Vallecillo, V. T. Vasconcelos, and A. Ravara. Typing the behavior of objects and components using session types. In *1st International Workshop on Foundations of Coordination Languages and Software Architectures (Foclasa 2002)*, volume 68, pages 439–456. ELSEVIER, March 2003. doi:10.1016/S1571-0661(05)80382-2. (cited at p. 42)

W. M. P. van der Aalst and T. Basten. Inheritance of workflows: an approach to tackling problems related to change. *Theoretical Computer Science*, 270(1-2):125–203, 2002. ISSN 0304-3975. doi:10.1016/S0304-3975(00)00321-2. (cited at pp. 42, 134, 144, 232, and 234)

W. van der Aalst and M. Weske. The P2P approach to interorganizational workflows. In *Proceedings of the 13th International Conference on Advanced Information Systems Engineering CAiSE '01*, volume 2068 of *LNCS*, pages 140–156. Springer, 2001. doi: 10.1007/3-540-45341-5_10. (cited at pp. 42, 234, and 272)

W. M. P. van der Aalst. The application of Petri nets to workflow management. *Journal of Circuits, Systems and Computers*, 8(1):21–66, 1998. doi:10.1142/S0218126698000043. (cited at p. 38)

W. M. P. van der Aalst, N. Lohmann, P. Massuthe, C. Stahl, and K. Wolf. From public views to private views – correctness-by-design for services. In M. Dumas and R. Heckel, editors, *Web Services and Formal Methods*, volume 4937 of *LNCS*, pages 139–153, Berlin / Heidelberg, 2008. Springer. ISBN 978-3-540-79229-1. doi:10.1007/978-3-540-79230-7_10. (cited at pp. 42, 144, 232, 234, and 272)

W. M. P. van der Aalst, A. J. Mooij, C. Stahl, and K. Wolf. Service interaction: Patterns, formalization, and analysis. In *Formal Methods for Web Services*, volume 5569 of *LNCS*, pages 42–88. Springer, Berlin / Heidelberg, 2009. ISBN 978-3-642-01917-3. doi:10.1007/978-3-642-01918-0_2. (cited at pp. 42, 144, and 232)

R. van Glabbeek. The linear time-branching time spectrum II. In E. Best, editor, *4th International Conference on Concurrency Theory Hildesheim, Germany, August 23-26, 1993 Proceedings CONCUR'93*, volume 715 of *LNCS*, pages 66–81. Springer, Berlin / Heidelberg, 1993. ISBN 978-3-540-57208-4. doi:10.1007/3-540-57208-2_6. (cited at p. 41)

R. van Glabbeek. The linear time-branching time spectrum I - the semantics of concrete, sequential processes. In *Handbook of Process Algebra, chapter 1*, pages 3–99. Elsevier, 2001. (cited at pp. 40 and 41)

K. Van Hee, N. Sidorova, and M. Voorhoeve. Soundness and separability of workflow nets in the stepwise refinement approach. In *Proceedings of the 24th international conference on Applications and theory of Petri nets*, ICATPN'03, pages 337–356, Berlin / Heidelberg, 2003. Springer. ISBN 3-540-40334-5. doi:10.1007/3-540-44919-1_22. (cited at pp. 28 and 39)

K. M. van Hee, A. J. Mooij, N. Sidorova, and J. M. van der Werf. Soundness-preserving refinements of service compositions. In *7th International Workshop, WS-FM 2010, Hoboken, NJ, USA, September 16-17, 2010. Revised Selected Paper*, LNCS, pages 131–145, Berlin / Heidelberg, 2011. Springer. ISBN 978-3-642-19588-4. doi:10.1007/978-3-642-19589-1_9. (cited at pp. 39 and 272)

V. T. Vasconcelos. Fundamentals of session types. In *9th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM 2009, Bertinoro, Italy, June 1-6, 2009, Advanced Lectures*, volume 5569 of *LNCS*, pages 158–186. Springer, Berlin / Heidelberg, 2009. ISBN 978-3-642-01917-3. doi:10.1007/978-3-642-01918-0_4. (cited at p. 42)

W. Vogler, C. Stahl, and R. Müller. A trace-based semantics for responsiveness. In J. Brandt and K. Heljanko, editors, *Proceedings of the 12th International Conference on Application of Concurrency to System Design, ACSD 2012, Hamburg, Germany,*

June 27-29, 2012, pages 42–51. IEEE, 2012. doi:10.1109/ACSD.2012.10. (cited at pp. 10, 39, and 42)

C. Wagner. Partner datenverarbeitender services. In M. Schwarick and M. Heiner, editors, *Proceedings of the 17th German Workshop on Algorithms and Tools for Petri Nets, AWPN 2010, Cottbus, Germany, October 07-08, 2010*, volume 643 of *CEUR Workshop Proceedings*, pages 154–159. CEUR-WS.org, October 2010. URL `http://ceur-ws.org/Vol-643/paper19.pdf`. (cited at p. 40)

C. Wagner. A data-centric approach to deadlock elimination in business processes. In D. Eichhorn, A. Koschmider, and H. Zhang, editors, *Proceedings of the 3rd Central-European Workshop on Services and their Composition, ZEUS 2011, Karlsruhe, Germany, February 21–22, 2011*, volume 705 of *CEUR Workshop Proceedings*, pages 104–111. CEUR-WS.org, 2011. URL `http://CEUR-WS.org/Vol-705/paper14.pdf`. (cited at p. 40)

D. Weinberg. Efficient controllability analysis of open nets. In R. Bruni and K. Wolf, editors, *Web Services and Formal Methods, Fifth International Workshop, WS-FM 2008, Milan, Italy, September 4–5, 2008, Proceedings*, volume 5387 of *LNCS*, Berlin / Heidelberg, September 2008. Springer. doi:10.1007/978-3-642-01364-5_14. (cited at pp. 40 and 271)

D. Weinberg. *Deciding Service Substitution - Termination guaranteed*. Dissertation, Universität Rostock, December 2012. ISBN 978-3843907385. (cited at pp. 39, 40, and 272)

K. Wolf. Reversing the construction of an operating guidelines. unpublished, 2007a. (cited at pp. 10 and 236)

K. Wolf. Generating Petri net state spaces. In *Petri Nets and Other Models of Concurrency – ICATPN 2007*, volume 4546 of *LNCS*, pages 29–42, Berlin / Heidelberg, 2007b. Springer. doi:10.1007/978-3-540-73094-1_5. (cited at pp. 43, 225, and 266)

K. Wolf. Does my service have partners? In *Transactions on Petri Nets and Other Models of Concurrency II*, volume 5460, pages 152–171, Berlin / Heidelberg, 2009a. Springer. doi:10.1007/978-3-642-00899-3_9. (cited at pp. 39, 40, 54, 55, 68, 69, 76, 81, 92, 96, and 272)

K. Wolf. A theory of service behavior. In O. Kopp and N. Lohmann, editors, *Proceedings of the 1st Central-European Workshop on Services and their Composition, ZEUS 2009, Stuttgart, Germany, March 2-3, 2009*, volume 438 of *CEUR Workshop Proceedings*, pages 1–7. CEUR-WS.org, March 2009b. URL `http://CEUR-WS.org/Vol-438/paper1.pdf`. (cited at p. 38)

K. Wolf, C. Stahl, J. Ott, and R. Danitz. Verifying deadlock- and livelock freedom in an soa scenario. *Application of Concurrency to System Design, International Conference on*, 0:168–177, 2009. ISSN 1550-4808. doi:10.1109/ACSD.2009.16. (cited at pp. 27, 39, and 40)

K. Wolf, C. Stahl, D. Weinberg, J. Ott, and R. Danitz. Guaranteeing weak termination in service discovery. *Fundamenta Informaticae*, 108(1-2):151–180, 2011. doi:10.3233/FI-2011-417. (cited at pp. 39 and 272)

A. Wombacher, P. Fankhauser, B. Mahleko, and E. Neuhold. Matchmaking for business processes based on choreographies. In *Proceedings of the 2004 IEEE International Conference on e-Technology, e-Commerce and e-Service (EEE'04)*, EEE '04, pages 359–368, Washington, DC, USA, 2004. IEEE Computer Society. ISBN 0-7695-2073-1. doi:10.1109/EEE.2004.1287334. (cited at pp. 40 and 46)

T. Yu, Y. Zhang, and K.-J. Lin. Efficient algorithms for web services selection with end-to-end qos constraints. *ACM Transactions on the Web (TWEB)*, 1, May 2007. ISSN 1559-1131. doi:10.1145/1232722.1232728. (cited at pp. 4 and 39)

J. M. Zaha, A. P. Barros, M. Dumas, and A. H. M. t. Hofstede. Let's dance: A language for service behavior modeling. In *On the Move to Meaningful Internet Systems 2006: CoopIS, DOA, GADA, and ODBASE*, volume 4275 of *LNCS*, pages 145–162, Berlin / Heidelberg, 2006. Springer. doi:10.1007/11914853_10. (cited at pp. 3 and 38)

B. Zhou, T. Yoneda, and B.-H. Schlingloff. Conformance and mirroring for timed asynchronous circuits. In *Proceedings of Asia and South Pacific Design Automation Conference 2001 (ASP-DAC2001)*, pages 341–346, 2000. (cited at pp. 42 and 150)

Acknowledgements

First of all, I would like to thank Wolfgang Reisig for an opportunity to start doing my Ph.D. with his research group in Berlin. I am grateful to Birgit Heene for her help and support on both administrative and non-administrative issues through the time of this thesis.

I owe my gratitude to Christian Stahl, Arjan Mooij, and Marc Voorhoeve. Their constructive advices and comments on the topic are most valuable especially when I had to make an outline and started writing the thesis. I thank Karsten Wolf for his insightful vision on the topic which could help me to structure my ideas and to understand better the work I was doing. I am also thankful to Holger Schligloff for his fruitful feedback on this thesis.

I appreciate financial support from METRIK (Modellbasierte Entwicklung von Technologien für selbstorganisierende dezentrale Informationssysteme im Katastrophenmanagement) graduate school, Humboldt-Universität zu Berlin. I thank Joachim Fischer for his confidence in my research topic and in me as a METRIK scholar.

I am thankful to my research fellows at the theory of programing group and METRIK graduate school in Berlin for driving a creative working atmosphere in both scientific and non-scientific related issues. I also thank my fellows at Burapha University Thailand for encouraging me to pursue my Ph.D. Special thanks to Tomkanok Chantarujirakorn for her wonderful support during my first years in Germany.

Last but not least, I am very grateful to my family in Thailand for their perpetual love and support. I express my gratitude to Vigerske family for fulfilling my family needs while I was in Germany. I thank Stefan for showing me in the recent years how to bring peace and happiness to my mind.

Jarungjit Parnjai
Berlin, April 2013